Titre: Title:	Verification of the Performance Properties of Embedded Streaming Applications via Constraint-Based Scheduling
Auteur: Author:	Olfat Elmahi
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Elmahi, O. (2016). Verification of the Performance Properties of Embedded Streaming Applications via Constraint-Based Scheduling [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2239/

UNIVERSITÉ DE MONTRÉAL

VERIFICATION OF THE PERFORMANCE PROPERTIES OF EMBEDDED STREAMING APPLICATIONS VIA CONSTRAINT-BASED SCHEDULING

OLFAT ELMAHI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
JUIN 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

VERIFICATION OF THE PERFORMANCE PROPERTIES OF EMBEDDED STREAMING APPLICATIONS VIA CONSTRAINT-BASED SCHEDULING

présentée par : ELMAHI Olfat
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

Mme BOUCHENEB Hanifa , Doctorat, présidente
M. PESANT Gilles, Ph. D., membre et directeur de recherche
Mme NICOLESCU Gabriela, Doctorat, membre et codirectrice de recherche
M. BELTRAME Giovanni, Ph. D., membre et codirecteur de recherche
M. BOIS Guy, Ph. D., membre
M. ZILIC Zeljko, Ph. D., membre externe

DEDICATION

To my daughters for bringing the hope and happiness to my life,
my husband for his everlasting love, support, and encouragements,
my parents and sisters for their unquestioning faith in me. . . .

ACKNOWLEDGEMENT

Firstly, I would like to thank my supervisors, **Gilles Pesant**, **Gabriela Nicolescu**, and **Giovanni Beltrame**, for their time, support, and wisdom. they have been not only providing me with invaluable ideas through out the work but also being available to help me every time I have challenges and difficulties.

I would like to give special thanks to **Tamer Abd El-Dayem**, my husband, who has been a great help not only during my PhD. study but also during the years of my life abroad.

I also like to thank my mother **Olfat Abou-Halawa** and my father **Ibrahim El-Mahi** for paying all the sacrifices to make me reach where I am today.

I extend my regards to all of my sisters **Merit E-Mahi**, **HebaTallah El-Mahi** and, **MenaTallah El-Mahi**, who have been encouraging me, supporting me, and more importantly praying for me. I deeply thank my aunt **Mervat Abou-Halawa** for her non-stop encouragement and support.

But above all, I thank Allah for providing me with such helpful professors, caring family, and supportive friends. **Thank You God !**

RÉSUMÉ

Les capacités et, en conséquence, la complexité de la conception de systèmes embarqués ont énormément augmenté ces dernières années, surfant sur la vague de la loi de Moore. Au contraire, le temps de mise en marché a diminué, ce qui oblige les concepteurs à faire face à certains défis, ce qui les poussent à adopter de nouvelles méthodes de conception pour accroître leur productivité. En réponse à ces nouvelles pressions, les systèmes modernes ont évolué vers des technologies multiprocesseurs sur puce. De nouvelles architectures sont apparues dans le multitraitement sur puce afin d'utiliser les énormes progrès des technologies de fabrication. Les systèmes multiprocesseurs sur puce (MPSoCs) ont été adoptés comme plates-formes appropriées pour l'exécution d'applications embarquées complexes.

Pour réduire le coût de la plate-forme matérielle, les applications partagent des ressources, ce qui peut entraîner des interférences dans le temps entre les applications dues à des conflits dans la demande des ressources. Les caractéristiques d'un SoC typique imposent de grands défis sur la vérification SoC à deux égards. Tout d'abord, la grande échelle de l'intégration du matériel mène à des interactions matériel-matériel sophistiquées. Puisqu'un SoC a de multiples composants, les interactions entre ceux-ci pourraient donner lieu à des propriétés émergentes qui ne sont pas présentes dans un seul composant. En second lieu, l'introduction de logiciels dans le comportement du matériel mène à des interactions matériel-logiciel sophistiqué. Puisqu'un SoC a au moins un processeur, le logiciel constitue une nouvelle dimension des comportements du SoC et donc apporte une nouvelle dimension à la vérification. Cela rend la vérification d'une tâche difficile, en particulier pour les applications de communication et de multimédia. Cela est dû à des contraintes non-fonctionnelles des modules matériel et logiciel, tels que la vitesse du processeur, la taille de la mémoire tampon, le budget de l'énergie, la politique de planification, et la combinaison de multiples applications.

Cette thèse préconise la programmation par contraintes (CP) comme un outil puissant pour la vérification des mesures de performance de MPSoCs. Dans ce travail, nous avons considéré des applications de diffusion sur l'architecture cible d'un système-sur-puce (MPSoC) multi-processeur comme un problème d'ordonnancement à base de contraintes. Nous l'avons testé séparément et en interaction avec d'autres types d'applications. L'idée est de créer un scénario au niveau du système qui prend en compte le flux de travail au niveau du système par rapport aux ressources du système et des exigences de performance, à savoir les délais

de la tâche, le temps de réponse, le CPU et l'utilisation de la mémoire, ainsi que la taille de la mémoire tampon. Plus précisément, nous examinons si le comportement des différentes interactions entre les composants du système d'exécution des tâches différentes peut être efficacement exprimé comme un problème d'ordonnancement à base de contraintes sur l'espace des entrées possibles du système, afin de déterminer si nous pouvons traiter des cas similaires d'échec en utilisant ce modèle. Résoudre ce problème consiste à trouver une meilleure façon d'inspecter le système en cours de vérification dans une phase de conception qui arrive très tôt et dans un délai beaucoup plus raisonnable.

Notre approche proposée a été testée avec diverses applications, différents flux d'entrée et des architectures différentes. Nous avons construit notre modèle en prenant en considération les architectures existantes sur le marché, des applications choisies qui sont en courante et comparé les résultats de notre modèle avec les résultats provenant de l'exécution des applications réelles sur le système. Les résultats montrent que la méthode permet de déterminer les conditions de défaillance du système dans une fraction du temps nécessaire à la vérification par simulation. Il donne à l'ingénieur d'essai la possibilité d'explorer l'espace de conception et d'en déduire la meilleure politique. Il contribue également à choisir une architecture appropriée pour des applications en cours d'exécution.

ABSTRACT

The abilities and, accordingly, the design complexity of embedded systems have expanded enormously in recent years, riding the wave of Moore's law. On the contrary, time to market has shrunk, forcing challenges onto designers who in turn, seek to adopt new design methods to increase their productivity. As a response to these new pressures, modern-day systems have moved towards on-chip multiprocessing technologies. New architectures have emerged in on-chip multiprocessing in order to utilize the tremendous advances of fabrication technology. Multiprocessor Systems on a Chip (MPSoCs) were adopted as suitable platforms for executing complex embedded applications.

To reduce the cost of the hardware platform, applications share resources, which may result in inter-application timing interference due to resource request conflicts. The features of a typical SoC impose great challenges on SoC verification in two respects. First, the large scale of hardware integration leads to sophisticated hardware-hardware interactions. Since a SoC has multiple components, the interactions between them could give rise to emerging properties that are not present in any single component. Second, the introduction of software into hardware behaviour leads to sophisticated hardware-software interactions. Since an SoC has at least one processor, software forms a new dimension of the SoC's behaviours and hence brings a new dimension to verification. This makes verification a challenging task, in particular for communication and multimedia applications. This is due to the non-functional constraints of hardware and software modules, such as processor speed, buffer size, energy budget, and scheduling policy, and the combination of multiple applications.

This thesis advocates Constraint Programming (CP) as a powerful tool for the verification of performance metrics of MPSoCs. In this work, we mapped streaming applications onto a target Multi-Processor System-on-Chip (MPSoC) architecture as a constraint-based scheduling problem. We tested it separately and in interaction with other application types. The idea is to create a system-level scenario that takes into account the system level work-flow with respect to System resources and performance requirements, namely task deadlines, response time, CPU and memory usage, and buffer size. Specifically, we investigate whether the behaviour of different interactions among system components executing different tasks can be effectively re-expressed as a constraint-based scheduling problem over the space of possible inputs to the system, finding if we can address similar cases of failure using this

model. Solving this problem means finding a better way to investigate and verify the System under verification in a very early design stage and in a much more reasonable time.

Our proposed approach was tested with various applications, different input streams and different architectures. We built our model for existing architectures on the market running chosen applications and compared our model results with the results coming from running the actual applications on the system. Results show that the methodology is able to identify system failure conditions in a fraction of the time needed by simulation-based verification. It gives the Test Engineer the ability to explore the design space and deduce the best policy. It also helps choose a proper architecture for the applications running.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENT iv

RÉSUMÉ v

ABSTRACT vii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVIATIONS xiv

CHAPTER 1 INTRODUCTION 1
- 1.1 Problem statement 3
- 1.2 Contributions 5
- 1.3 Organization 6

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW 8
- 2.1 Embedded System Verification 8
  - 2.1.1 Verification Technology 8
  - 2.1.2 System-Level Verification Challenges 11
  - 2.1.3 Current Research on System-Level Verification 14
- 2.2 Constraint Programming 16
  - 2.2.1 Basic Concepts of Constraint programming 16
  - 2.2.2 Constraint Programming and System-Level Verification 18
  - 2.2.3 IBM CPLEX Optimizer 20

CHAPTER 3 MOTIVATIONAL EXAMPLES 29
- 3.1 Motivative Examples One and Two 30
  - 3.1.1 Streaming Applications For Synthesis Case 31
  - 3.1.2 Platform Architecture For Synthesis Case 34
- 3.2 Motivative Example Three 36

3.2.1 Streaming Applications For Industrial Case . . . 36
3.2.2 Platform Architecture For Industrial Case . . . 38

CHAPTER 4 MAPPING PACKET FLOW OF STREAMING APPLICATIONS ONTO MPSOC . . . 44
4.1 Constraint-Based Scheduling Approach . . . 44
4.1.1 Stream model . . . 45
4.1.2 Decision Variables . . . 48
4.1.3 Constraints . . . 48
4.2 Experimental Results . . . 55

CHAPTER 5 MAPPING FRAME FLOW OF STREAMING APPLICATIONS ONTO MPSOC . . . 58
5.1 Alternative Model . . . 58
5.1.1 Stream model . . . 58
5.1.2 Decision Variables . . . 59
5.1.3 Constraints . . . 59
5.2 Experimental Results . . . 64

CHAPTER 6 MAPPING TASKS FLOW OF STREAMING APPLICATIONS ONTO MPSOC . . . 68
6.1 Industrial-Case Model . . . 68
6.1.1 Stream model . . . 68
6.1.2 Decision Variables . . . 73
6.1.3 Constraints . . . 75
6.2 Experimental Results . . . 90

CHAPTER 7 CONCLUSION . . . 106
7.1 Work Summary . . . 106
7.2 Future Work . . . 107

REFERENCES . . . 109

LIST OF TABLES

Table 3.1	Sitara Processor Features	43
Table 3.2	Summary of BeagleBone components used in the CSP model described in Chapter 6 [12]	43
Table 4.1	Design space for the experimental platform	55
Table 4.2	experimental Results: the 2^{nd} line indicate PE size, 3^{rd} line Determine application specifications, and 4^{th} line is Bus Delay. note the results with f symbol indicate that solution found in less then 15 second	55
Table 5.1	Design space for the experimental platform	65
Table 6.1	Details of Streaming Application Running on the Industrial Platform	71
Table 6.2	Devices input and traffic calculation for the different applications running in the system	74
Table 6.3	Single case running test results. * indicates further explanation in the results discussion.	94
Table 6.4	different case combinations running test results. * indicates further explanation in the results discussion.	97
Table 6.5	Summary of the Different Parameters Considered for Each Application	100

LIST OF FIGURES

Figure 1.1 Design and Verification Gaps. Design productivity growth continues to remain lower than complexity growth - but this time around, it is verification time, not design time, that poses the challenge. A recent statistic showed that 60-70% of the entire product cycle for a complex logic chip is dedicated to verification tasks [25]. 1

Figure 1.2 the verification gap from simulation point of view. Simulation is the main approach to design verification, and there are simulation platforms suitable for different abstraction levels. However, as integration level increases, simulation efficiency always decreases; during the requirement to thorough simulation increases. There is a widening gap between the required and available simulation performance [35]. 2

Figure 1.3 the structure of the thesis. 2

Figure 2.1 Example 1: Framework Example. 10

Figure 2.2 HW/SO design trade-off. 18

Figure 2.3 Typical use of CP optimizer [13]. 20

Figure 2.4 Elementary cumul function expressions([13]). 25

Figure 2.5 Example of alternative constraint in ILOG solver. 27

Figure 3.1 MPEG-4 and VOIP packet flow in MPSoC architecture. 30

Figure 3.2 A 2 x 2 regular mesh MPSoC architecture. 31

Figure 3.3 MPEG-4 frame dependencies (The arrows display the dependants between frames decompressed.) 32

Figure 3.4 **Upper picture:** MPEG-4 and VOIP packet flow in MPSoC architecture - **Lower picture:** Application-specific topologies under test. IP denotes processor cores, pr private-Memories, and NI Network interface. 35

Figure 3.5 BeagleBone Black Considered flow. 38

Figure 3.6 BeagleBone Black board ([12]). 39

Figure 3.7 BeagleBone Black block digram. 41

Figure 4.1 Original and decompressed packets in the system floe chart 46

Figure 4.2 Capacity constraints 50

Figure 4.3 Alternative tasks constraints 50

Figure 4.4 MPEG-4 Group Of Picture order and frame dependencies 53

Figure 5.1 MPEG-4 Group Of Picture order and frame dependencies 62

Figure 5.2 Results for MPEG4 and VOIP separately 65
Figure 5.3 Results for MPEG4 and VOIP combined 66
Figure 6.1 streaming applications flow in MPSoC industrial architecture. 69
Figure 6.2 DS-5 streaming Counters used to measure system performance. 72
Figure 6.3 Streamline Ds-5 Dhrystone application. 91
Figure 6.4 Streamline DS-5 WhestStone application. 92
Figure 6.5 Streamline for ice weasel applications browsing multiple web pages in a separate window. 96
Figure 6.6 Streamline for ice weasel application browsing multiple web pages in a separate tab. 96
Figure 6.7 Two different streamline analysis for MPlayer applications running local video with low decompression rate for newsletter program (Case $b(c2)$). 101
Figure 6.8 Two different streamline analysis for MPlayer application running local video with high decompression rate for football game (Case $a(c1)$). . . 102
Figure 6.9 Two different streamline analysis for MPlayer application running live stream video with high decompression rate for football game (Case $g(c1)$). 103

LIST OF ABBREVIATIONS

API	Application Programming Interface
ATPG	Automatic Test-Pattern generator
BDD	Binary Decision Diagram
BW	Band Width
CP	Constraint Programming
CSP	constraint satisfaction problem
CPU	Central Processing Unit
DUT	Design Under Test
DSP	Digital Signal Processing
FSM	Finite-State-Machine
FIFO	Frst In First Out
GOP	Group of Pictures
HW	Hardware
HDL	Hardware Description Language
HVL	Hardware Verification Languages
IC	Integrated Circuit
IDE	Integrated Development Environment
I/O	Input/Output
IP	Internet Protocol
MPEG	Moving Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
MP	Mathematical Programming
NOC	Network-on-Chip
OOP	Object Oriented Programming
OR	Operations Research
OPL	Optimization Programming Language
PCI	Peripheral Component Interconnect
PAPS	Periodic Admissible Parallel Schedules
PE	Processing Element
RT	Real Time
SOC	System On Chip
SW	Software
SDF	Synchronous Data Flow

TB	Testbench
TDMA	Time Division Multiple Access
VLSI	Very-Large-Scale Integration
VOIP	Voice Over Internet Protocol
WCET	Worst Cases Execution Time

CHAPTER 1 INTRODUCTION

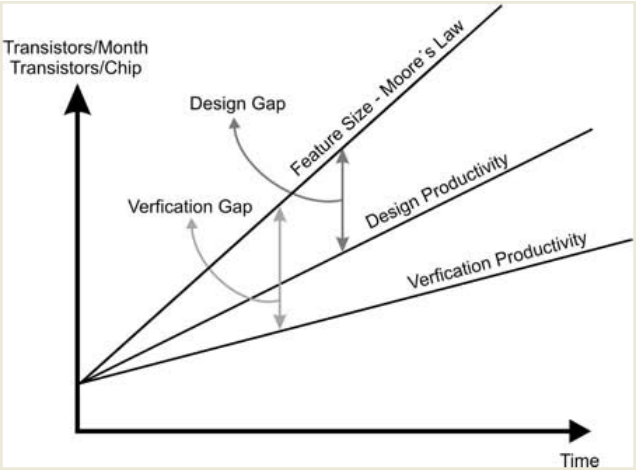

Figure 1.1 Design and Verification Gaps. Design productivity growth continues to remain lower than complexity growth - but this time around, it is verification time, not design time, that poses the challenge. A recent statistic showed that 60-70% of the entire product cycle for a complex logic chip is dedicated to verification tasks [25].

The multiprocessor System-on-Chip (MPSoC) designs have become a very popular choice for modern embedded systems [41]. These designs use complex on-chip networks to integrate different programmable processor cores, specialized memories, and other components on a single chip. The parallel nature of MPSoCs makes verification a challenging task, in particular for communication and multimedia applications. This is due to the non-functional constraints of hardware and software modules, such as processor speed, buffer size, energy budget, and scheduling policy [45], the combination of multiple applications.

System-level design and verification methodologies such as Constraint Programming (CP) have been introduced as a solution to handle the design complexity of embedded systems [45]. The power of CP comes from the fact that validity, quality, and test specification requirements for any system are naturally modeled through constraints, which are naturally represented as a Constraint Satisfaction Problem (CSP). In this work, we introduce a constraint-based scheduling model for concurrent streaming applications on MPSoCs with and without considering processor scheduling policies.

Our aim is to identify the critical system parameters (e.g. buffer size) that can lead to unsatis-

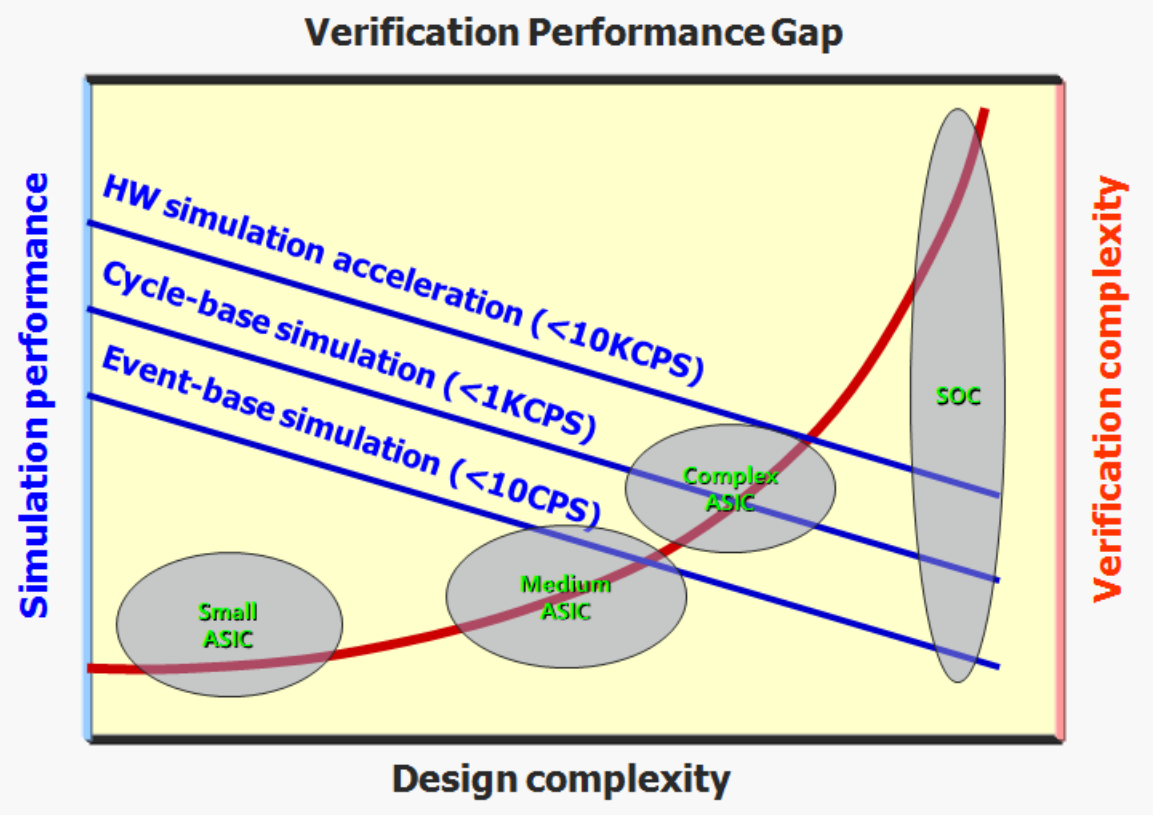

Figure 1.2 the verification gap from simulation point of view. Simulation is the main approach to design verification, and there are simulation platforms suitable for different abstraction levels. However, as integration level increases, simulation efficiency always decreases; during the requirement to thorough simulation increases. There is a widening gap between the required and available simulation performance [35].

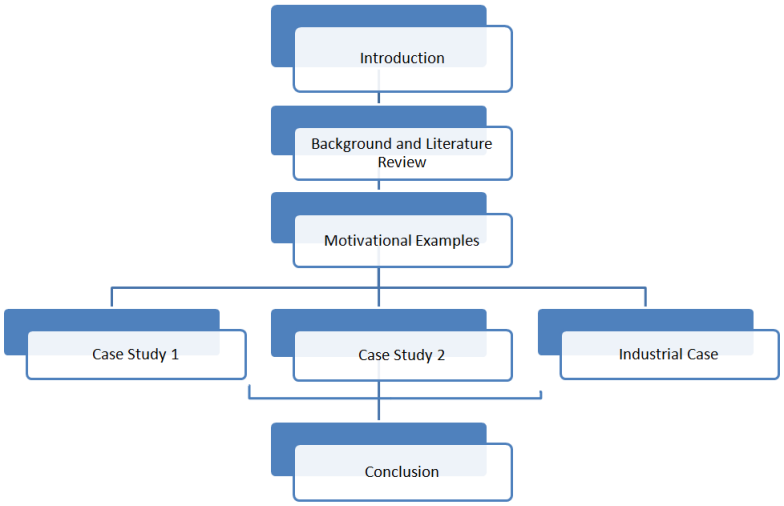

Figure 1.3 the structure of the thesis.

fied application constraints. Also, we propose design optimization (e.g. buffer minimization) to increase the system efficiency and reduce its cost.

1.1 Problem statement

From the design complexity point of view, although the SoC paradigm is very beneficial—it has practically reduced designing a complex system to integrating pre-designed and reusable components – the verification of the SoC becomes the critical bottleneck in further improving SoC design productivity [40]. Generally speaking, verification refers to the practice of detecting errors in designs. Designs that are not thoroughly verified are not worth manufacturing; and errors should be corrected as early as possible—correcting errors at a late stage could be forbiddingly costly.

Verification was regarded as the subservient issue compared with the implementation of a design. This view soon became invalid. The well-known Moore's law suggests that the complexity of integrated circuits is growing at an exponential rate against time, whereas multiple sources claim the verification complexity is growing at a double-exponential rate, i.e., exponential with respect to Moore's law. Figure 1.1 and 1.2 illustrate the growing *verification gap* between the integrated circuit (IC) verification capability and the IC design and manufacture capabilities. Nowadays about 50%-80% of the design time and efforts are spent in verification. It has become well known that verification is the main bottleneck in integrated circuit design [33].

The features of a typical SoC impose great challenges on SoC verification in two respects.

– First, the large scale of hardware integration leads to sophisticated hardware-hardware interactions. Since a SoC has multiple components, the interactions between them could give rise to emerging properties that are not present in any single component.
– Second, the introduction of software into hardware behavior leads to sophisticated hardware-software interactions. Since a SoC has at least one processor, software forms a new dimension of the SoC behaviors and hence brings a new dimension in verification.

This is why our research focuses on the process of test-case generation based on two different directions.

– Application-Dependent Verification. The application, instead of the test bench, should

take the more active role of test-case control, especially parallelism management; the test bench, not the test-program, should take the relatively passive observation role.
– Interaction-Oriented Verification. The object-under-test should be the interactions among components, rather than the components themselves.

The main approach to verifying a design, especially a very complex one, is by simulation. Simulation is so important to verification that the term *simulation* and *verification* largely share the same meaning in practice. *Simulation* refers to the practice of running tests on models of a design before the design is actually manufactured. The term *model* refers to a presentation of the hardware under design in the form of software. The simulation approach inherently has the *simulation performance issue*. That is, simulation is a very time-consuming process, while VLSI designers are constantly under the time-to-market pressure. Fast and accurate simulation is always desired; however, being fast and being accurate are always competing metrics for simulation-based approaches. The *verification gap* viewed from the simulation point of view is shown in Figure 1.1 and 1.2.

As designs are becoming more complex, the requirement of thorough verification is soaring, whereas the performance of various simulation technologies is degrading. The simulation performance issue is more outstanding for SoC verification due to its high level of integration. Due to the high design complexity and manufacturing cost, new system-level design methodologies for embedded systems have emerged to deal with the increasing time-to-market pressure.

At system-level, the *concurrency* or *parallelism*, among multiple components is the defining characteristics of a hardware system. The concurrency forms a new verification dimension. System-level bugs are usually discovered in *corner cases* where parallel processes interact with each other in an unexpected way. *Resource-competition* is an inevitable consequence of concurrency. Hardware components could show functional problems when competing with each other for resources, even if they have already passed component-level verification. Listed next are some potential bugs found at system-level:

– Interaction between blocks that are assumed verified.
– Conflict in accessing shared resources.
– Arbitration problems and missing deadline.
– Priority conflicts in tasks.

– Unexpected hardware/software sequences.

All these bugs are related to component-to-component interactions, especially to concurrent interactions with resource competitions. Therefore the key to system-level verification is to construct concurrency/resource-competition satisfactorily.

1.2 Contributions

The main contribution of this thesis is improved verification and exploration of system-level concurrency in MPSoCs. It addresses Constraint Programming (CP) as a powerful tool for the verification of performance metrics of MPSoCs.

We studied the possibility of creating a system-level scenario that takes into account the system level work-flow with respect to System resources and performance requirements, namely task deadlines, response time, CPU and memory usage, and buffer size. Specifically, we investigate whether the behavior of different interactions among system components executing different tasks can be effectively re-expressed as a constraint-based scheduling problem over the space of possible inputs to the system, finding if we can address similar cases of failure using this model. Solving this problem means finding a better way to investigate and verify the System under verification in address a certain case of failure in a very early design stage and in a much more reasonable time.

The idea is to choose various applications with different input streams and different work-flow architectures, study its performance requirements with different order and interactions, and then see how it affects the system. Having this information helps defining a set of constraints to represent how each application should work on any architecture. This will be used in creating different system-level scenarios. It gives the ability to explore different architectures, detect possible system failure at an early stage, and in some cases even suggest a proper solution. Note that this approach is not about the detection of bugs in the logic of the applications.

Our proposed approach was tested with various applications, different input streams and different architectures. Results show that the methodology is able to identify system failure conditions in a fraction of the time needed by simulation-based verification. It gives the Test

Engineer the ability to explore the design space and deduce the best policy, also it helps choose the proper a recommended architecture for the applications running. We built our model for already built architectures in the market running chosen applications and compare our model results with the results coming from running the actual applications on the system. The research has produced the following publications:

– First Paper [16]: this paper contains the work introduced in Chapter 4. The main contribution of this work is to provide a technique to map a synthesised but still interesting MPSoC as a constraint scheduling problem and to generate interesting test cases based on streaming applications. These tests are capable of discovering corner cases that would cause system failure. The introduced model has a limitation of scaling. It has a large number of tasks, because it uses packets generated by the applications to represent its flow in the system.

– Second Paper [17]: this paper contains the work introduced in Chapter 5. This work resolved the scaling problem we faced in the first model. Here we introduced a new way to decrease the number of tasks representing each application running in the system while respecting the same system and application constraints. To achieve this, we changed the way the application is mapped in the system from packets to frames. This proposal improved the model performance, covering more test cases, and running the application for longer periods of time on the system which make the results more accurate.

– Third Paper (pending): this paper contains the work introduced in Chapter 6. Here we deal with an industrial case study with a commercial application. We overcome the problem of representing hundreds of millions of instructions as a limited number of tasks running in the system. These tasks calculations are based on traffic generated by its applications. Also, we introduced new applications and studied how their overlapping with stream applications did affect the system.

1.3 Organization

The main structure of the thesis is shown in Figure1.3

– Chapter 1: gives an introduction to the thesis, including the problem statement, contribution and structure.

- Chapter 2: lays a firm background for further development of the thesis. The main topics include:
  - Basic concepts give a brief explanation to Constraint Programming and Verification Technology focusing on the system level verification.
  - Literature review of System-Level verification
  - IBM CPLEX Optimizer the tool used in this thesis to create our model. It includes a brief explanation of Scheduling with IBM ILOG CPLEX Studio and IBM ILOG CPLEX Studio Search strategy which is the default strategy we used.
- Chapter 3 gives a description of MPSoC architecture platforms and applications considered in building the first two case studies described in Chapters 4 and 5. And the specifications of the industrial platform and applications used with the third case study in Chapter 6.
- Chapters 4, 5 and, 6 have the explanation and discussion of the proposed constraint programming model for each case study in two different sections:
  - Constraint-Based Scheduling Approach: contains an explanation for the stream model and its decision variables followed by a description of each of the constraints and why we added them.
  - Experimental results with a complete description and discussion of the model results. The strengths and weaknesses are also discussed.
- Chapter 7 concludes the thesis.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

With the increasing complexity in embedded products and the improvements in development technology, Multi-Processor System-On-Chip (MPSoC) architectures have become widespread. They can now be found in many complex real-time systems (e.g. cell phones, video processing or avionics). Theses systems usually share the same set of applications with a common well-characterized context. However, each possible set of applications that can be active concurrently in an MPSoC platform leads to a different use-case. And each use-case has to be verified and tested while meeting several additional design constraints (e.g. energy consumption or time-to-market). Therefore it takes a great amount of time for these applications to be tested and optimized and, mechanisms to efficiently explore the different possible HW-SW design interactions in complete MPSoC systems are in great need. In this chapter we try to give a little bit of history of what have been done before and how this was affected by the work in this thesis.

As discussed in Chapter 1, we want to tackle the two major problems: the generation of test cases to verify the system and the interaction between the system components when different tasks are running on the platform. This chapter is divided into two parts: To start with, we discuss basics of embedded-system verification, system-level verification problems and its literature review. Because Constraint Programming is considered an important part of system-level verification researchers, the second part of this chapter focusses on Constraint Programming basics, approaches and, researchers on the generation of test cases for verification of mixed software/hardware systems.

2.1 Embedded System Verification

2.1.1 Verification Technology

The goal of verification is to ensure that the design meets the functional requirements as defined in the functional specification. Verification of SOC devices takes anywhere from 40 to 70 percent of the total development effort for the design. Some of the issues that arise are how much verification is enough, what strategies and technology options to use for verification, and how to plan for and minimize verification time. These issues challenge both verification engineers and verification solution providers.

A wide variety of verification technology options are available within the industry. These options can be broadly categorized into two classifications [42]

– **Simulation-based methods or dynamic verification.** In this category, the verification engineers develop a set of tests known as test-cases to stress a given design. Hence, the design is often called design-under-test or DUT. A test-case could be a very abstract description of a scenario the DUT should be exercised in.

– **Formal methods or static verification.** This category is called static since no tests are needed. Instead, the verification engineer should provide design properties (the properties a correct design should have) in the form of temporal logic. The design is represented in Finite State Machine (FSM) form. Then a Binary Decision Diagram (BDD) based model-checking tool computes whether the design abides by the properties. If one property is violated, the tool will produce at least one counter example – a sequence of input to the FSM that leads to the violation of that property.

While formal verification techniques have their clear advantages, most notably the ability to formally prove functional correctness of the design, they can hardly cope with modern complex designs at the level of a single unit or larger. To this end, simulation-based verification, in which the design behavior is checked by simulating it over external inputs, accounts for roughly ninety percent of the overall verification efforts and resources. This is why the work in this thesis focuses on Simulation-based Verification and to be more specific on system level verification.

Simulation-based Verification and Stimuli Generation

The essence of simulation-based (or dynamic) verification is to test how the design conducts itself when confronted with challenging stimuli. Stimuli generation, in turn, deals with the problem of creating the appropriate stimuli in order to test the DUT as thoroughly as possible. The nature and abstract level of the stimuli depends on the object being tested and the level of verification. Automatic Test-Pattern Generator (ATPG) tools [37] test the manufacturing of circuits by applying sequences of lowest-level bit-vectors at the circuit's input interfaces. A full processor can be tested by generating test programs in the assembly language of the processor. At the highest abstraction level, system level stimuli can include commands that

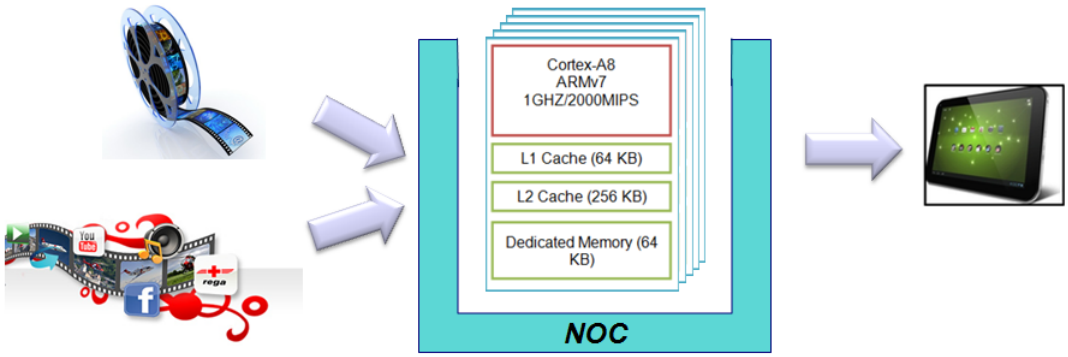

Figure 2.1 Example 1: Framework Example.

produce transactions involving multiple system devices.

A good stimulus first needs to be valid with respect to the requirements imposed by the DUT. It should also be of high quality in the sense that it tests the behaviour of the DUT in some desired circumstances to improve the coverage of the tested behaviour, reach challenging corner cases, and hopefully trigger a bug. Also, the stimuli had better be able to expose the bug if it indeed occurs during the test execution (and not render it unobservable, for example, by masking its effects).

The most basic, technology-free method for generating stimuli is to write them by hand. Surprisingly enough, this is still being done, especially if there are only a few simple directed stimuli that are needed, or when there is no available technology to generate the type of stimuli required. Needless to say, this method is limited in capacity, expensive, error-prone, and often cannot achieve the precise stimuli that are required. A technology for automatic stimuli generation is therefore needed.

The generated stimuli, usually in the form of test programs, are designed to trigger architecture and micro-architecture events defined by a verification plan [20]. The input for a test program generator is a specification of a test template. An example of such a test template would be a set of tests that exercise the data cache of the processor and that are formed by a series of double-word store and load instructions. The generator produces a large number of distinct well-distributed test program instances that comply with the user's specification. The variation among different instances is achieved through a large number of random decisions made during the generation process. In addition, generated test programs must meet

two inherent requirements: (1) tests must be valid, that is, their behaviour should be well defined by the specification of the verified system; (2) test programs should also be of high quality, in the sense that they should expand the coverage of the verified system and focus on potential bugs. Recently, technology has shifted toward constraint-based formulations of the generation task and generation schemes driven by solving constraint satisfaction problems (CSPs) [20].

2.1.2 System-Level Verification Challenges

At system level, the concurrency among components is the defining characteristic of the hardware system. Its bugs are usually discovered in corner cases where parallel processes interfere with each other in an unexpected way. Resource competition is an inevitable consequence of concurrency. HW components could show functional problems when competing with each other for resources, even if they have already passed component-level verification.
System Level Stimuli Generation has four challenges:
– Specifying system level scenarios in an abstract form while generating the required low level stimuli
– Generating coordinated system-level stimuli projected to each and every core in the system
– Effectively handling configuration changes (e.g.: 2-way system vs. 8-way system)
– Adapting to core modifications and new cores (e.g.: PCI to PCIe)

Currently, neither formal methods nor general simulation-based approaches are dealing with the system-level behaviours such as concurrency/resource-competition satisfactorily.

Formal Methods

Formal methods are simply not in the position to discover system-level bugs due to the nature of these bugs, so the industry is depending less on formal methods [18].

– Bugs caused by implementation details: system-level bugs could arise from an inaccurate or a miss-interpreted design specification, as well as from the detailed implementation of that specification. Formal methods may suit well for the former, in which implementation details could be abstracted away. However, if the design is represented as an FSM with implementation details, the model-checkers will not scale up.

– Control and data-intensive failures: system-level bugs often arise in scenarios in which data-intensive and control-intensive behaviours are loosely intertwined; whereas formal methods work best with control-intensive applications.

– Failures across components: it is often impossible to attribute a system-level bug to a particular hardware component; instead, the bug may be caused by the ill-matched behaviors of multiple components [24]. It will be very difficult and un-scalable for formal methods to deal with combined or communicating FSMs.

More importantly, the fact that the user is responsible to provide properties to formal tools is the fundamental barrier to applying formal methods on system-level verification. Hardware systems, which are made of multiple components and may be represented with implementation details, do not have fixed failure modes. Therefore, verification engineers are constantly faced with the difficult choice of "expecting something unexpected". As a consequence, they cannot postulate those properties that they are yet to know.

Testbench based Simulation Generation

System-level verification essentially based on the simulation approach, in which tests are applied to the design-under-test (DUT) via a structure called testbench (TB). Yet, the current practices based on testbench (TB) construction have encountered some problems:

– **The TB stimulates DUT and observes the response from the exterior of the DUT**. This arise from the distinction between external and internal behaviours where the TB treats the DUT as a black box. It applies stimulation and observation from the outside of a DUT, so it is naturally difficult to force the TB to control and observe the DUT's internal behaviours. There are white-box approaches, i.e., adding control and observation points around components inside a DUT, to supplement the black-box approach. However, we could argue that this approach is still black-box natured in the sense that the similar control and observation issues still exist at component level.

– **The divergence between the techniques to develop DUTs and those to develop TBs rapidly becoming two distinct entities.** In a word, a TB is more a software occurrence in the real world than a hardware structure in the simulated

world ([42]).

  - The languages used to develop a DUT continues to be HDLs. In addition, for precise simulation, the DUT should be described in the synthesisable subset of HDL constructs. A DUT is largely understood as a hardware structure in the simulated world.

  - The languages used to develop a TB migrate to HVLs and other object-oriented programming (OOP) languages. These languages provide dynamic constructs to make possible dynamic connections. Still, being dynamic also means the loss of synthesisability. The state-of-the-art TBs require OOP paradigm or even beyond.

- Increasing TB complexity. When components are integrated into a system, new capabilities have to be added in the TB to test the emerging properties caused by the integration. In this way TB complexity could grow faster than DUT complexity. This will eventually prevent us from relying on TB alone to verify a more complex DUT.

As we can see, the Testbench based Simulation Generation creates serious complications for system-level verification. It is very complex to take control and observation responsibilities; but in the end, it still does not touch the main challenging task of test generation.

Software.

Software (SW) may be responsible for the majority of the SoC functionalities, yet, software does not have a proper place in TB-Based verification methodologies.

- SW-based verification is naturally used in processor verification. In this case, SW is organised at the instruction-level and usually targets the micro-architecture of the processor-under-test. Therefore, this category of verification methodologies does not apply to system-level verification.

- SW-based tests are also found in SoC manufacturing-testing. Since the driving force of design verification and manufacturing testing are substantially different, those methods shed limited light on the area of SoC verification.

– The idea of "HW/SW co-verification" is practiced as running an operating system (OS) and application software on a SoC model for the purpose of software verification. Therefore, running these software components is the "liability" rather than the "asset" for the hardware team.

– SW in the form of hardware diagnostics programs could be interpreted as the "asset" to SoC verification. However, these diagnostics (a) are either too simplistic or too specific, and (b) are poorly automated and require manual development. So using this form of software cannot serve as a major verification approach.

Software is the valid testing factor alongside the DUT and the TB. For an SoC DUT, it is common practice for a verification engineer to write tests in the form of software snippets. This common practice actually demonstrates the software's capabilities in controlling and observing a DUT. Although writing test cases in software is often treated as an ad-hoc verification technique, we should realise that the introduction of software in hardware verification has overturned the traditional TB-Based verification problems.

2.1.3 Current Research on System-Level Verification

Simulation-based verification is a well-known method to determine the response time of embedded systems. Simulation is the process of mimicking key characteristics of a system or process. It can be performed at different levels of abstraction. At one end of the spectrum, one finds tools such as Wind River Simics ([36]) or ReSP ([6]) which simulate a complete system (software and hardware) in detail. Such simulators are used for low-level debugging or for hardware/software co-design. This type of simulation can trade off speed and accuracy: it can yield accurate timing analysis with long simulation time, or focus on speed by limiting its scope to functional simulation.

At the other end of the spectrum we find scheduling simulators, which abstract from the actual behaviour of the system and only analyze the scheduling of the system's tasks, specified by key scheduling attributes and execution times ([42]).

System verification technology has recently shifted towards the use of Constraint Programming (CP) for random functional test generation. For example, several constraint-based generators were developed at IBM: X-Gen [19] for system-level verification, GenesysPE [1] at

the architecture level, Piparazzi [2] at microarchitecture level, FPGen [4] and DeepTrans [3] for hardware units, and SoCVer [31] for SoCs. These works use constraints both to describe the hardware system and to express which areas of the design should be tested. They also use randomness to achieve a balanced distribution of the generated test data in these areas. Results show this relatively new trend as a promising alternative to simulation-based verification of complex hardware systems.

Systems handling stream applications like MPEG-4 or VOIP, for example, define a pipeline work flow with strict ordering of data transfers between system components. Such systems typically employ a single controller core and a specific software model. Verification of these systems requires creating a system-level scenario that takes into account the system level workflow [31, 8, 34], with some random variance allowed.

Such systems are also decoupled, and allow a large variability in the interactions between the cores, supporting a large number of system configurations. It is important to verify the conjunction of functionalities of the different components, since errors are usually triggered by specific interactions. Some errors can only be exposed if the components interact locally in time and space, that is, if the interaction involves using the same resources at the same time. One also has to consider multiple system resources such as CPUs, disks, and network links that require coordinated scheduling to meet the end-to-end performance requirements of streaming applications.

In general, scheduling problems are computationally challenging, and have been subject of active research in Constraint Programming (CP) and in Operations Research (OR) for many years [30].

Constraint programming has been used more specifically in the scheduling of task graphs on MPSoCs without violating computation capacity and communication bandwidth [7, 24], and for data-stream (or cyclic) scheduling [10]. Since [24, 10] proposes a global cumulative constraint for cyclic scheduling problems, they are not really applicable to our case. In our work we consider problems arising from different tasks scheduling needs and interleaving between tasks applications at different timing. On the other handv [7] is more related. They tackle the problem of allocating and scheduling processors, communication channels and memories of multicore platform. They compare different approaches and results show that Constraint Programming is a proper tool for dealing with multi-task applications achieving

very good performance.

Verification of embedded streaming applications in communication and multimedia domains on MPSoCs has been widely explored by using the Synchronous Data Flow (SDF) model [29, 38]. Lee and Messerschmitt [29] first present general techniques to construct periodic admissible parallel schedules (PAPS) on a limited number of multiprocessors. Govindarajan et al. [21] propose a linear programming formulation to obtain maximal throughput and minimized buffer cost for SDF models without computation (number of processors) constraints. Eles et al. [18] first address the scheduling on distributed systems with communication protocols optimization. Stuijk [38] propose a mapping and TDMA/list scheduling design flow for throughput constrained SDF applications on MPSoCs. Zhu [44] propose a design optimization framework for adaptive real-time streaming applications based on reconfigurable devices. They further investigate buffer minimization and task scheduling issues for streaming applications in [45]. In our work we took advantages of CSP to validate scheduling of embedded streaming applications on distributed systems. We used objective function to propose design optimization (e.g. buffer minimization - number of processors). We studied the effect can be caused by interaction between this applications and other type of applications. Or, study the behaviour of the same application on different conditions (e.g. different frame rate or size).

We can see a big part of research on system level verification lately shifted toward Constraint Programming. For the problem of allocating and scheduling tasks on MPSoCs, which is the problem we target in this research, the major advantage of using CP is the clarity and understandability of the models. CP modeling is more flexible than other methods like the Mixed Integer Programming (MIP) on many challenging optimization problems, including mapping and scheduling [14].In the next few sections we will give some details on CP and how it fits in with system level verification for MPSoC running stream applications.

2.2 Constraint Programming

2.2.1 Basic Concepts of Constraint programming

Constraint Programming (CP) deals with modelling and solving CSPs. CSPs are mathematical problems defined as a set of variables each with its own domain and whose state must satisfy a number of constraints or limitations constraints that restrict the values those variables can presume ([32]). For example, in an task/resource-scheduling problem, the vari-

ables may be the start and end time of each task on a certain resource, and a constraint may specify the maximum number of tasks can be run on one resource at the same time, or tasks sequence and dependant (one task can not be start before another end).

Mathematical formalism

Mathematically, a CSP P is a triplet (V, D, C) consisting of a set of variables V, a corresponding set of domains D, and a set of constraints C. A solution to a CSP is an assignment of a value to each variable out of the domain of the variable such that all constraints are satisfied. A CSP is satisfiable if it has at least one solution and unsatisfiable otherwise.

In the task/resource-scheduling example, assuming there are N tasks, we would have $2N$ variables: one start time variable and one end time variable for each task. The domain of the two variables may be the list of available hours for each of tasks period. Constraints may specify such things as deadline for any particular task, requirements on resource sizes, and tasks dependency. Mathematically, constraints are known as relations. A relation on a set of k variables is the list of all legal combinations of k values, each taken from the domain of the corresponding variables. For example, consider three variables a, b, c, with domains {1, 2}, {1, 2, 3}, {1, 2, 3}, respectively. A constraint requiring that the three variables assume different values may be represented by the mathematical relation {(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1) }

CSP modelling

CSP modelling is the process of translating a real-world constraint problem into a CSP. It involves identifying the variables in the problem, the variable domains (i.e., the values each variable can have before considering conflicts due to the constraints), and the constraints. There is usually more than one way to choose the variables and domains. For example, in the task/resource-scheduling problem, we could have chosen the variables to be all combinations of start-end dates of a task. Under this choice, the domains of all variables may be the names of the tasks plus "*null*," signifying that no task is scheduled at this particular resource in a certain time.

2.2.2 Constraint Programming and System-Level Verification

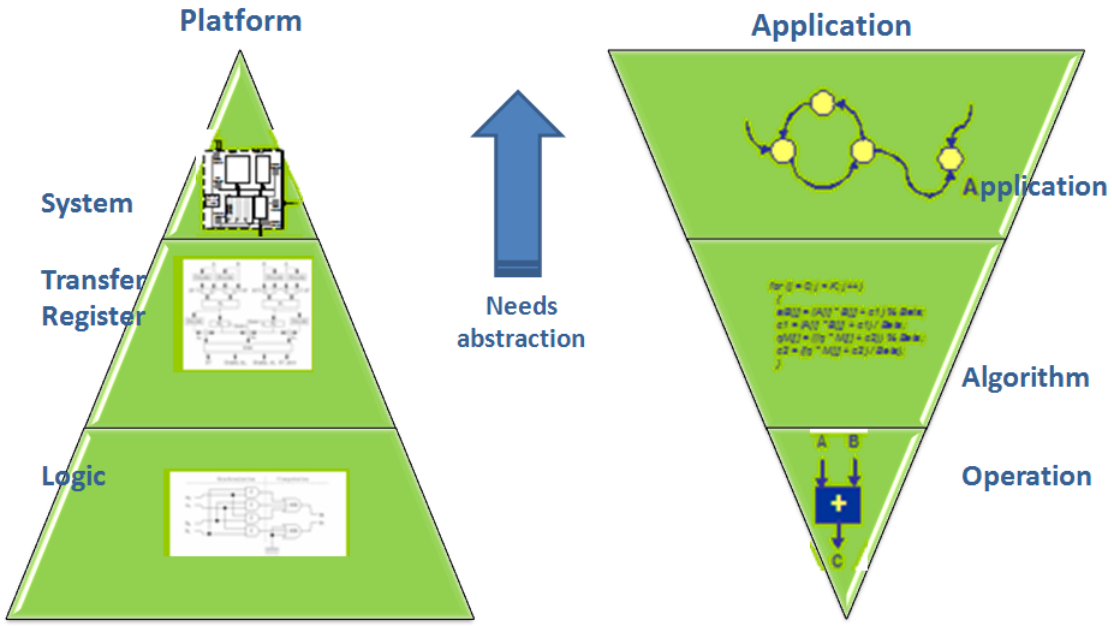

Figure 2.2 HW/SO design trade-off.

Validity, quality, and test specification requirements are naturally modelled through constraints . Consider the example, Figure 2.1, for testing performance of concurrent streaming applications running on MPSoCs via simulation-based verification. Such tests have three main challenges which are:

1. creating inputs, or 'stimuli' that are:
   - valid according to the hardware specification and the simulation environment,
   - interesting in the sense that they are likely to excite prone-to-bugs areas of the design, and
   - diverse.
2. Find the proper Trade-Offs in time, behavior and area in HW/SW design (Figure 2.2) with the proper abstraction level expressing both the complexity of applications and execution platforms

3. Analyze performance (ex: WCET "Worst Case Execution Time", Hard/Soft tasks deadlines, resources limit, ...).

Item (1) is dealt with by modelling the entire hardware specification, as well as that of the simulation environment, as a set of mandatory constraints over the simulated variables (memory addresses, data transferred, processor instruction parameters, and so on). Item (2) is dealt with in two ways: first, generic expert knowledge is modelled as a set of soft, non-mandatory, constraints (for example, a soft constraint may require the result of operation a + b to be zero, because this is a known prone-to-bugs area of the floating point processing unit). Item (3) is dealt with by adding a target for the model to achieve like Minimize memory used or delays. In addition, the verification engineer, who is directly responsible for creating the stimuli, may add mandatory and non-mandatory constraints to any particular run, directing the stimuli into required scenarios.

Going back to Figure 2.2, having a stream application running on MPSoC. We have item (1) as processes execute in a data driven manner, and communicate with each other via FIFO channels as a mandatory constraint. And item (2) to decide the proper process size to be considered (i.e. frames, packets, ...). Item (3) can be an optimization issue such as the rate of displaced stream frames per second may be maximized or the buffer used minimized. Once all specifications are modelled, this set of mandatory and non-mandatory constraints can be fed into a constraint solver, which comes up with a solution to the constraint problem in the form of a valid and interesting stimulus. In order to achieve item (3), the solver typically has a built-in diversification mechanism, for example, that several frames access the same cache block, thus causing contention on resources shared between different processors. For a CSP to drive test program generation, the program, or its building blocks, should be modelled as constraint networks. A random test program generator can, therefore, be viewed as a CSP solver. It constructs a CSP from the user requirements and the system model, and produces a large number of distinct program instances that satisfy the constraints.

Constraint satisfaction problems that represent test programs share several characteristics. Test program generation requires random, well-distributed solutions over the solution space [20, 9], as opposed to the traditional requirement of reaching a single solution, all solutions, or a "best" solution [27]. Huge domains are the result of large address spaces in modern architectures. The combination of huge domains (e.g., 264 values), linear constraints

(e.g., $a = b + c$) and non-linear non-monotonic constraints (e.g., $A = B \oplus C$, where A, B, and C are bit vectors, and $\oplus$ is the bit-wise XOR operation) make storing and operating on these domains a difficult task. Other characteristics include a hierarchy of hard and soft constraints and dynamic modelling (i.e., new variables being "born" when values are assigned to other variables).

2.2.3 IBM CPLEX Optimizer

(CPLEX Optimization Studio supports the rapid development, deployment and maintenance of mathematical programming(MP) and constraint programming (CP) models from a powerful integrated development environment (IDE) built on the Optimization Programming Language (OPL), through programmatic APIs, or alternatively through third-party modeling environments [26].) IBM ILOG CPLEX CP Optimizer is a constraint programming (CP)

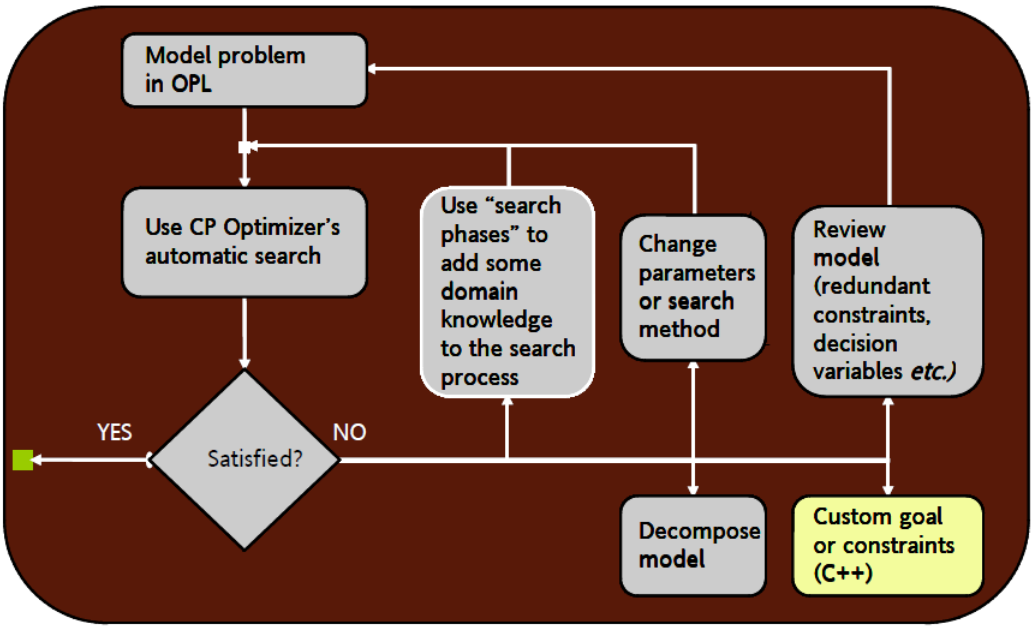

Figure 2.3 Typical use of CP optimizer [13].

optimizer for solving scheduling problems, it also help solving some combinatorial optimization problems that cannot be linearized and solved simply using traditional mathematical programming methods (see Figure 2.3).

Scheduling with IBM ILOG CPLEX Studio

The CPLEX Optimization Studio provides easy and efficient access to CPLEX CP Optimizer features, which are specially chosen to solve detailed scheduling problems over fine-grained time. There are, for example, keywords mainly considered to represent such aspects as tasks and temporal constraints. It offers a workbench of modeling features in the CPLEX CP Optimizer engine that intuitively and naturally tackle the issues inherent in detailed scheduling problems from manufacturing, construction, driver scheduling, and more.

In a detailed scheduling problem, the most basic activity is assigning start and end times to an interval (interval here represent a task need to be scheduled in the system). The CPLEX CP Optimizer implementation is particularly useful for fine-grained scheduling. Scheduling problems also involve the management of minimal or maximal capacity constraints for resources over time and of alternative modes to perform a task.

A typical scheduling problem is defined by:

- A set of time interval definitions of activities, operations, or tasks to be completed, that might be optional or mandatory.
- A set of temporal constraints definitions of possible relationships between the start and end times of the intervals.
- A set of specialized constraints definitions of the complex relationships on a set of intervals due to the state and finite capacity of resources.
- A cost function: for instance, the time required to perform a set of tasks, cost for some optional tasks that are not executed or the penalty costs of delivering some tasks past a due date.

A scheduling model has the same format as other models in OPL:

- Data structure declarations.
- Decision variable declarations.
- Objective function.

– Constraint declarations.

OPL provides specialized variables, constraints and keywords designed for modeling scheduling problems.

Data structure declarations

Data declarations allow you to name your data so that you can reference it easily in your model. For example, if your data in a table defines the traffic running of one application at one system resource, you might want to call your item of data $trafic_{ij}$ where i=1,..., noApps, j=1,..., noResources, where noApps is the number of applications in your model and noResources is the number of system resources. You tell OPL that your model uses this data by declaring:

$$int \;\; noApps = ...;$$
$$int \;\; noResources = ...;$$
$$float \;\; trafic[1..noApps][1..noResources] = ...;$$

Decision variable declarations

Variable declarations name and define the type of each variable in the model. For example, if you want to create a variable that equals the capacity of each resource used in the system, you can create a variable named $capacity_i$ where j=1,..., noResources:

$$dvar \;\; int+ \;\; capacity[1..noResources];$$

The dvar keyword indicates that you are declaring a decision variable. Since int+ indicates that the variables are nonnegative, this statement declares an array of nonnegative integer variables.

For scheduling there are specific additional decision variables, namely **interval:** In OPL, activities, operations and tasks are represented as interval decision variables. An in-

terval has a start, an end, a length, and a size. An interval variable allows for these values to be variable within the model. The start is the lower endpoint of the interval and the end is the upper endpoint of the interval. By default, the size is equal to the length, which is the difference between the end and the start of the interval. In general, the size is a lower bound on the length. Also, an interval variable may be optional, and whether or not an interval is present in the solution is represented by a decision variable. If an interval is not present in the solution, this means that any constraints on this interval acts like the interval is "not there." The exact semantics will depend on the specific constraint.

$$dvar\ \ interval\ \ app1[tinTask][rinResourc]\ \ size\ \ Duration[t]\ \ optional;$$

Other types exists but we did not use it in the scope of this thesis.

Objective function

The objective function is an expression that you want to optimize. This function must consist of variables and data that you have declared earlier in the model. We use objective function in our model to optimize our system. For example: determine the minimum cache size that can be used to successfully run the system. Or, get the minimum delay that can be used.

$$minimize\ \ sum(i\ \ in\ \ 1..noApps)\ \ startOf(app[i][1]);$$
$$minimize\ \ sizOf(capacity[1]);$$

Constraint declarations

Precedence constraints Precedence constraints are common scheduling constraints used to restrict the relative position of interval variables in a solution. These constraints are used to specify when one interval variable must start or end with respect to the start or end time of another interval. A delay, fixed or variable, can be included. For example, if I have two different intervals a and b to be scheduled on a number of resources. Different precedence

constraints can be expressed in OPL.

$$dvarintervala;$$
$$dvarintervalb;$$

List of precedence constraints in OPL:

– **endBeforeStart:** if we wants interval a ends before the start of interval b with at least x time units.

$$endBeforeStart(a, b, x);$$

– **startBeforeEnd:** if we wants interval a starts before the end of interval b with at least x time units.

$$startBeforeEnd(a, b, x);$$

– **endAtStart:** if we wants interval a ends at exactly the same time interval b starts.

$$endAtStart(a, b);$$

– **endAtEnd:** if we wants interval a ends at exactly the same time interval b ends.

$$endAtEnd(a, b);$$

– **startAtStart:** if we wants interval a starts at exactly the same time interval b starts.

$$startAtStart(a, b);$$

– **startAtEnd:** if we wants interval a starts at exactly the same time interval b ends.

$$startAtEnd(a, b);$$

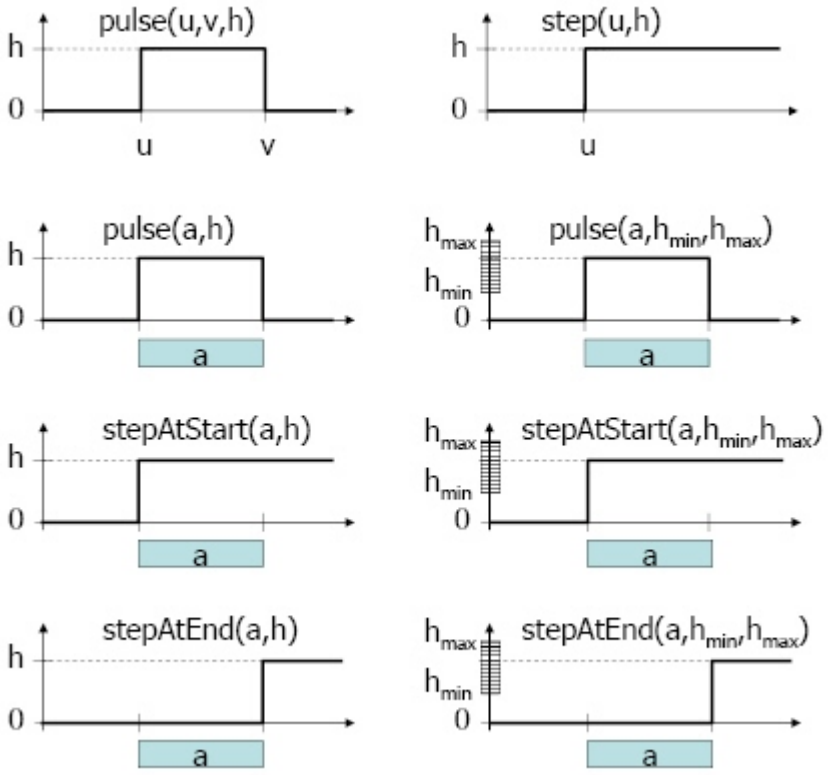

Figure 2.4 Elementary cumul function expressions([13]).

Cumulative constraints A cumulative function expression, represented in IBM ILOG OPL by cumulFunction, can be used to model a resource usage function over time. This function can be computed as a sum of interval variable demands on a resource over time. An interval usually increases the cumulated resource usage function at its start time and decreases it when it releases the resource at its end time (pulse function).

For resources that can be produced and consumed by activities (for instance the contents of an inventory or a tank), the resource level can also be described as a function of time. A production activity will increase the resource level at the start or end time of the activity whereas a consuming activity will decrease it. The cumulated contribution of activities on the resource can be represented by a function of time, and constraints can be modeled on this function (for instance, a maximal or a safety level).

The value of the expression at any given moment is constrained to be nonnegative. A cumulative function expression can be modified with the atomic demand keywords (Figure 2.4):

- step, which increases or decreases the level of the function by a given amount at a given time;
- pulse, which increases or decreases the level of the function by a given amount for the length of a given interval variable or fixed intervals;
- stepAtStart, which increases or decreases the level of the function by a given amount at the start of a given interval variable;
- stepAtEnd, which increases or decreases the level of the function by a given amount at the end of a given interval variable.

A cumulative function expression can be constrained to model limited resource capacity by constraining that the function be less than or equal the capacity [13].

Example of step functions

There is an interval A, fixed in time. Interval A increases the level of the resource by x time units at the start of the interval, modeled by applying stepAtStart, created with Interval A and the value x, to the cumulative function:

$$cumulFunction \quad ff = \quad stepAtStart(A, \quad x);$$

A more simpler example is to consider a function measuring a consumable resource. The level of the resource is zero, until time 2 when the value is increased to 4. This is modeled by modifying the cumulative function with the elementary cumulative function step at time 2:

$$cumulFunction \quad ff = \quad step(2, \quad 4);$$

No overlap constraints To constrain the intervals in a sequence such that they:

- Are ordered in time corresponding to the order in the sequence.
- Do not overlap.
- Respect transition times

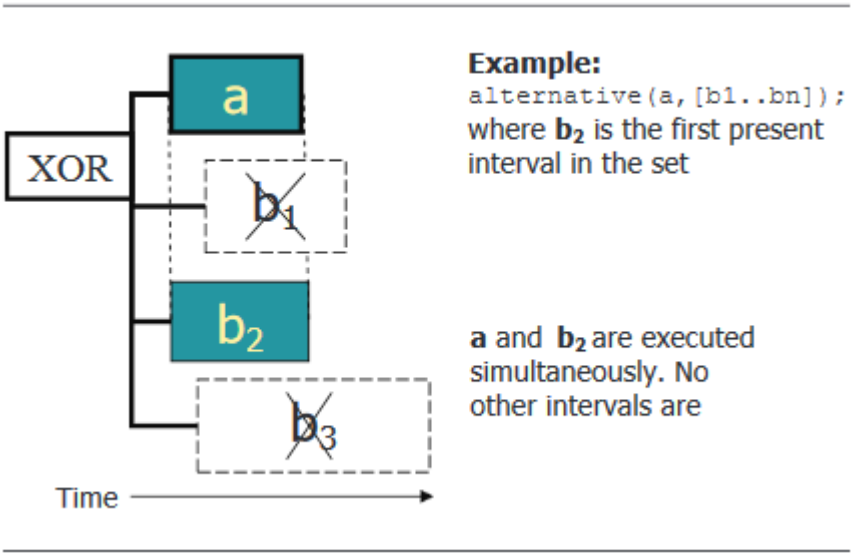

Figure 2.5 Example of alternative constraint in ILOG solver.

Alternative constraints An alternative constraint between an interval decision variable a and a set of interval decision variables B states that interval a is executed if and only if exactly one of the members of B is executed. In that case, the two tasks are synchronized. That is, interval a starts together with an interval from set B and ends together with it (Figure 2.5).

This type of constraint used in our system to represent the alternative resources that can be used by an application. Other types exists but we did not use them in the scope of this thesis.

IBM ILOG CPLEX Studio Search Strategy

Scheduling can be viewed either as a constraint satisfaction problem or as a constraint optimization problem. When we think of scheduling as a constraint satisfaction problem, our aim is to find a schedule that satisfies the constraints, whatever they may be. When we think of scheduling as an optimization problem, our aim is to find a schedule that is optimal or close to optimal with respect to a given optimization criterion. The optimization criteria usually relate to time, capacity, and sequence: typically the makespan, tardiness, peak capacity, or transition cost.

In our model we used the default search strategies recommended by IBM ILOG CPLEX scheduler. Its automatic search combines Large Neighbourhood Search with a portfolio of neighbourhoods and completion strategies together with Machine Learning techniques to converge on the most efficient neighbourhoods and completion strategies for the problem being solved [28]

CHAPTER 3 MOTIVATIONAL EXAMPLES

In system level design, it is essential to capture the functional behaviour and architectural characteristics independently for performance analysis and design space exploration. The resources we need to allocate and schedule in MPSoCs are heterogeneous: they include processing (micro-processors, DSPs, hardware accelerators), storage (i.e. on-chip memories) and communication (i.e. on-chip buses and I/Os) elements.

The embedded system requirements for distributed embedded systems (DES) can be divided into different groups which each impose a number of constraints on the scheduling problem. The functional behaviour of a embedded system is determined by the tasks and resources that constitute the system. Typical functional behaviour requirements are those that control task execution order or task allocation, that is, how a task should execute. An embedded system also have temporal behaviour requirements in addition to the functional ones. The temporal behaviour of a task depends mainly on the environment (sensors, actuators or other tasks) that the task interacts with, that is, when a task should execute. These requirements directly affect the modelling of the application tasks and consequently the construction of the scheduling constraints.

Apart from mere software requirements, it is also a practical consideration that the development of embedded systems is made cost-effective so as to allow for mass-production of the system. That is why development using off-the-shelf hardware components has become a viable alternative in modern designs. Other practical aspects of embedded system design encompass the introduction of weight and power-consumption requirements. This means that cost, performance and various physical characteristics of the hardware components (processors, memory and buses) need to be conveyed to the constraint construction process ([15]).

In this thesis, we have developed a scheduling framework based on constraint programming which attempts to tackle these needs. We mapped streaming applications onto a target Multi-Processor System-on-Chip (MPSoC) architecture as a constraint-based scheduling problem. We introduced three motivated examples discussed in Chapters 4, 5 and, 6. The first two examples discuss mapping streaming applications on a synthesis MPSoC system architecture in two different ways. The third example discussed mapping streaming applications on a industrial MPSoC system architecture. Further more, introduced more

applications into the system. This chapter introduces preliminaries of the streaming application, the architecture platforms, and a declarative constraint programming paradigm used in this thesis.

3.1 Motivative Examples One and Two

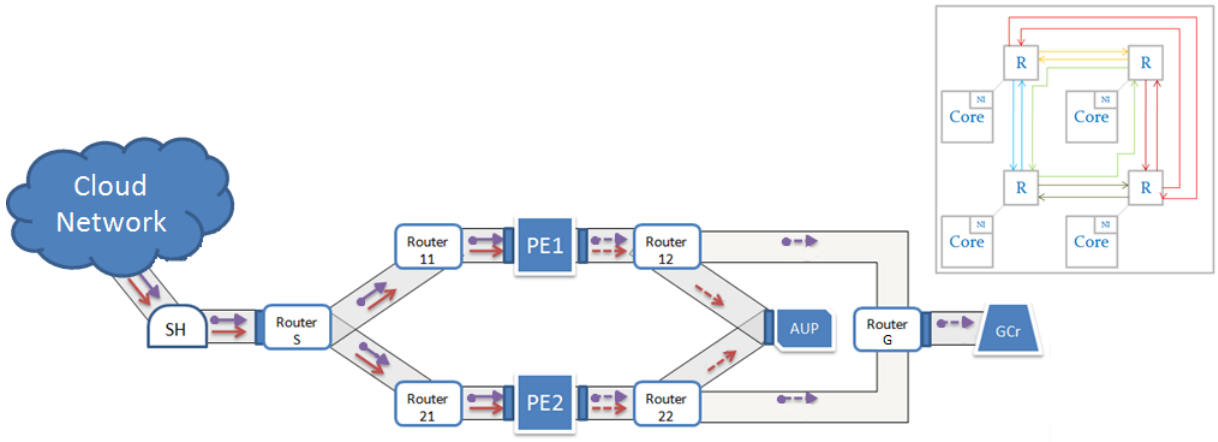

Figure 3.1 MPEG-4 and VOIP packet flow in MPSoC architecture.

These two examples considered MPEG-4 and VOIP as the two streaming applications mapped into the DUT (see Figure 3.1). Both applications have a significant impact on the traffic through the system. They have a bounded flow control with synchronization restriction and, minimum accepted performance. This is why it can be used in creating a complex scenarios used to discover interesting corner cases detected onto the DUT.

For the DUT we consider the regular 2-D mesh Network On Chip (NoC). It is simple but can be scaled easily (Figure 3.2).

As a case study, the MPEG4 was tested with five different standard frame sizes (QCIF, SDTV, HDTV, HDTV1, and HDTV2) where frameSize $=$ pixelDepth $\times$ Width $\times$ Height. The VOIP application was tested in two different cases: a non-restricted delay case which represents applications with buffering flexibility (such as messaging), and a restricted delay case for applications with hard deadlines and quality assurance.

The inputs to the system are packeted trace files for both applications. This makes the

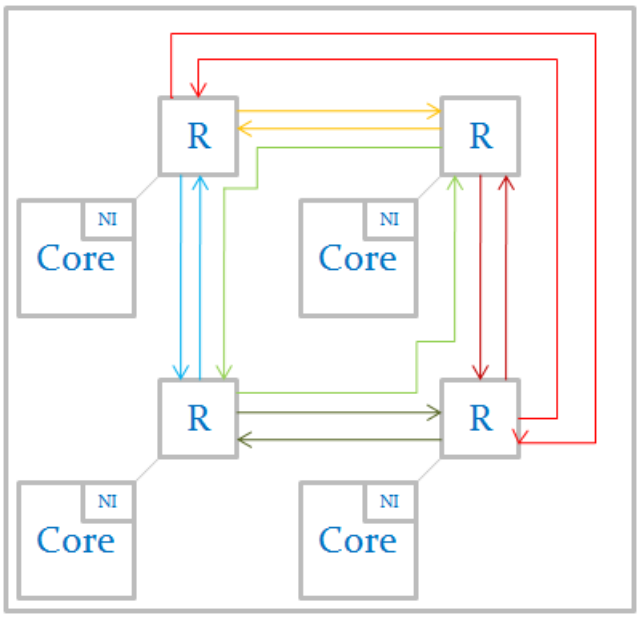

Figure 3.2 A 2 x 2 regular mesh MPSoC architecture.

application more realistic and allows the use of different streams with different data rates.

In the following paragraphs we will give a brief description of the nature of each application and constrains it forces. The details of the DUT and how it was represented in the CSP model. A detailed discussion for each application will be provided in the following chapters.

3.1.1 Streaming Applications For Synthesis Case

MPEG-4:

Our first application is MPEG-4. The MPEG standard defines four distinct pictures encoding: Intra-coded Picture (I-Picture), Predictive-Coded Picture (P-Picture), Bidirectional-Predictive-Coded Picture (B-Picture) and DC-Coded Picture (D-Picture). The I-Picture is coded using information only from the picture itself. The P-Picture is coded using motion compensation prediction in reference to a previous I-Picture or another P-Picture. B-Picture is coded in reference to either previous or future I-Pictures or P-Pictures. Finally, the D-Picture stores the DC component of each DCT block. The I, B and P pictures are arranged in a periodic pattern known as a Group of Pictures (GOP). Figure 3.3 shows the GOP of MPEG-4 video that we used, and the relationships among pictures. The MPEG-4 Group of

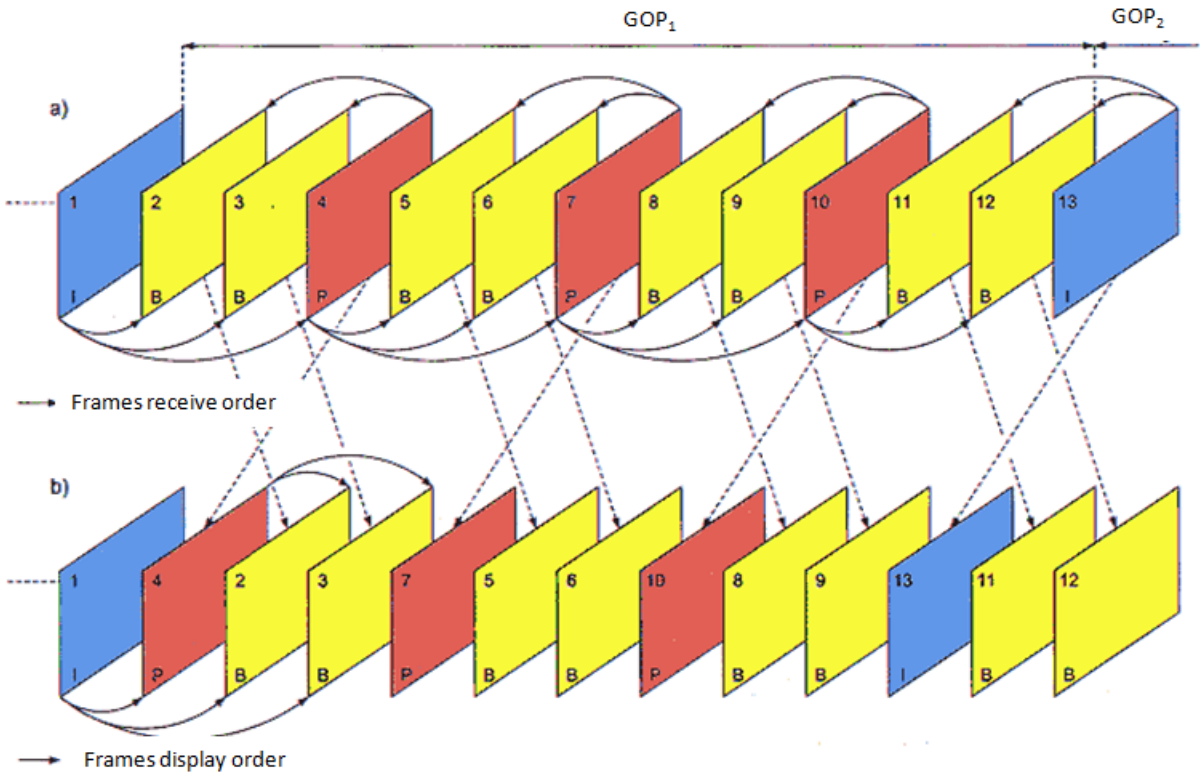

Figure 3.3 MPEG-4 frame dependencies (The arrows display the dependants between frames decompressed.)

Pictures (GOP) is made of 12 frames in the following order: IBBPBBPBBPBB ([23]).

The first two B pictures (2 and 3) are bi-directionally coded using the past frame (I frame 1) and the future frame P (frame 4). Therefore, each B picture is encoded based on the previous and following I and/or P pictures. P pictures on the other hand are dependent of previous I or P pictures. It is worth mentioning that due to these dependencies the decoding order will be different from the encoding order. The P frame 4 must be decoded before B frames 2 and 3, and I frame 1 (the last I frame) before B frames 11 and 12. If the MPEG-4 sequence is transmitted over the network, the actual transmission order should be 1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 1, 11, and 12. From the explanation above about the dependencies, it is easy to conclude that I-Pictures are the most important ones since they contain the actual video content and all other pictures are error-coded based on the I frames.

In this application the main constraints that should be satisfied are the following:

- The input data stream should be in the received order but the display stream must be in the decoding order.
- The number of frames per second displayed should be at least 30 fps.
- The output file should be bigger than the input file.

VOIP:

VoIP is a technique for transmitting voice data over the Internet [5]. When sending voice traffic over IP networks, a number of factors contribute to overall voice quality as perceived by an end user. Two of the most important factors are end-to-end delay in the voice carrier path and degraded voice quality. Performance requirements of VoIP by mapping the human perceived voice quality is based on the two network centric parameters:

- Packet loss: It was reported in the literature [11] that voice packet loss rates are acceptable within 1-3 % and the quality becomes intolerable when more than 3% of the voice packets are lost.

- Maximum Tolerable Delay: one-way transmission time for connections with adequately controlled echo should be in the 0-150 ms range to be acceptable for most user applications. Because end-to-end delay for VoIP depends on various components of the packet network, the queuing delay is variable depending on hops the

voice packet travelled through. Based on experiments proposed in [11] we assume the queuing delay to be at most 5 ms.

3.1.2 Platform Architecture For Synthesis Case

Networks-on-Chip (NoC) are a promising solution for designing scalable communication architectures for MPSoC, featuring better modularity and design predictability when compared to bus based systems. For this reason we will consider the NoC verification as the architecture of our synthesis case study.

In the synthesis case, we give a demonstration of a SoC Figure 3.4 to map our strategy on. We consider the regular 2-D mesh Network On Chip (NoC). We considered this architecture according to its application packet flow shown in Figure 3.1.

The system architecture consists of: one shared memory (SH) acting as a receiver for different application streams, two processing elements (PE1, PE2) each with its own private memory each to generate the output stream, one frame buffer for video display (GCr), and an audio output port (AuP) for audio display. All connected via 2x2 router to handle traffics between different modules. This architecture may be considered very simple but it can be easily scaled to more complex architecture.

In the CSP model sometimes we represent the same resource with two different symbols when the traffic goes through this resource more than once. In Figure 3.1, we have two different parts. The upper one shows the traffic flow through the system starting from being revived till it gets processed. It is numbered from 1 to 13. the lower part of the figure show the system topology and were is the correspondent traffic flow on it. The symbols used in the CSP model to express routers between resources are the same as the one used it the figure. it is explained as follows:

- RS = Router at SH's output.
- R11 = Router at PE1's input.
- R12 = Router at PE1's output.
- R21 = Router at PE2's input.
- R22 = Router at PE2's output.
- RG = Router at the GCr's input.

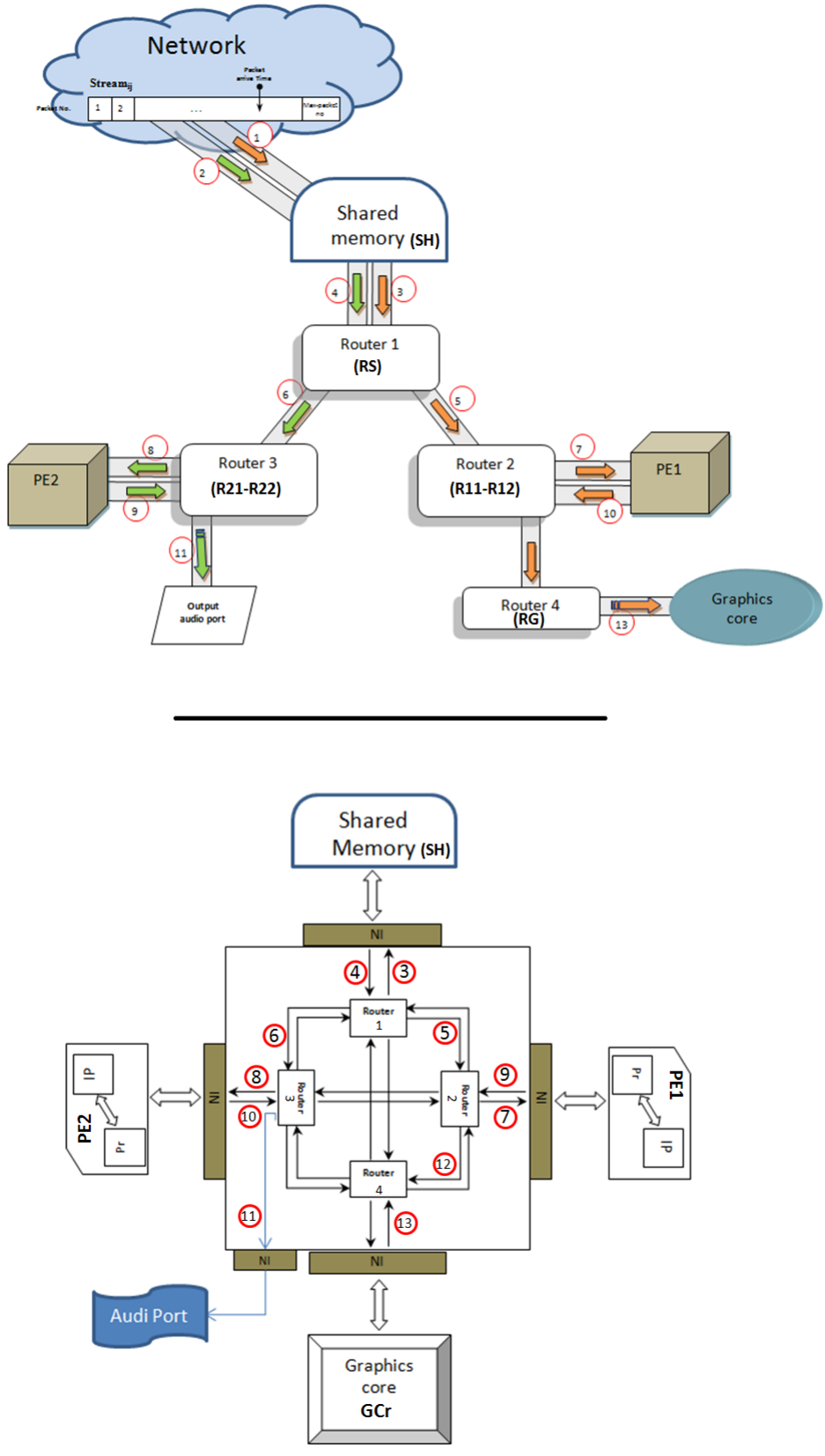

Figure 3.4 **Upper picture:** MPEG-4 and VOIP packet flow in MPSoC architecture - **Lower picture:** Application-specific topologies under test. IP denotes processor cores, pr private-Memories, and NI Network interface.

Assume that every IP of a SoC represents different application traffic and respects its real-time constraints. The architecture/micro-architecture definition and verification with respect to the SoC performance must then focus on the following critical components:

- Communication structures;
- Shared memory controllers.

3.2 Motivative Example Three

This example considered *MPlayer* a commercial application to represent MPEG-4 and VOIP streaming applications mapped into the DUT. Along with this application we added another four applications to introduce more load in the system. This causes more interesting scenarios. For the DUT we chooses BeagleBone Black.

In the following paragraphs we will give a brief description of the nature of each application and the constraints it forces, the details of the DUT and how it was represented in the CSP model. A detailed discussion for each application will be provided in the following chapters.

3.2.1 Streaming Applications For Industrial Case

Five applications were used to represent batch, interactive and real-time computations on the BeagleBone system to measure the system performance and create test cases:

- Dhrystone(batch)— Developed by Reinhold Weicker in 1984. This benchmark is used to measure and compare the performance of computers. The test focuses on string handling, as there are no floating point operations. It is heavily influenced by hardware and software design, compiler and linker options, code optimization, cache memory, wait states, and integer data types.

- WhestStone (batch)— This test measures the speed and efficiency of floating-point operations. It contains several modules that are meant to represent a mix of operations typically performed in scientific applications. A wide variety of C functions

including sin, cos, sqrt, exp, and log are used as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. This test measure both integer and floating-point arithmetic.

- MPlayer (Real-Time/Cashed)— is a popular movie player for GNU/Linux. It has support for most video and audio formats and is thus highly versatile, even if it is mostly used for viewing videos. It is a time stamp-based system capable of playing synchronized audio and video streams. It adapts to its system environment by adjusting the quality of playback based on the system load. For the experiment, we use this application in two different modes:

  - News(Real-Time/Cashed)— This application displays synchronized audio and video streams from website/local disk. Each media stream flows under the direction of an independent thread of control. The audio and video threads communicate through a shared memory region and use timestamps to synchronize the display of the media streams. The video input stream contains frames at 24bpp format at 30 frames/second with different frame size. The audio input stream contains standard 44100Hz 2ch floatle (4 bytes per sample) samples. The captured data is from a CBC news network. It represents a low frame change rate.

  - Entertain(Real-Time/Cashed)— This application displays synchronized audio and video streams from website/local disk. Each media stream flows under the direction of an independent thread of control. The audio and video threads communicate through a shared memory region and use timestamps to synchronize the display of the media streams. The video input stream contains frames at 24bpp format at 30 frames/second with different frame size. The audio input stream contains standard 44100Hz 2ch floatle (4 bytes per sample) samples. The captured data is from a you-tube website contains a mix of television programming. It represents a high frame change rate.

  - Music(Real-Time/Cashed)— This application displays audio streams from website/local disk. Each media stream flows under the direction of an independent thread of control. The audio input stream contains standard 44100Hz 2ch floatle (4 bytes per sample) samples.

- Iceweasle (interactive) — is a free software rebranding of the Mozilla Firefox web

browser distributed by the GNU Project. It is compatible with Linux, Windows, Android and OS X. The GNU Project keeps IceCat in synchronization with upstream development of Firefox while removing all trademarked artwork. It also maintains a large list of free software plug-ins. In addition, it features a few security features not found in the mainline Firefox browser.

– Wget (Real-Time) — is a computer program that retrieves content from web servers, and is part of the GNU Project. Its name is derived from World Wide Web and get. It supports downloading via HTTP, HTTPS, and FTP protocols. We used it here to test the effect of file downloads on other internet activities.

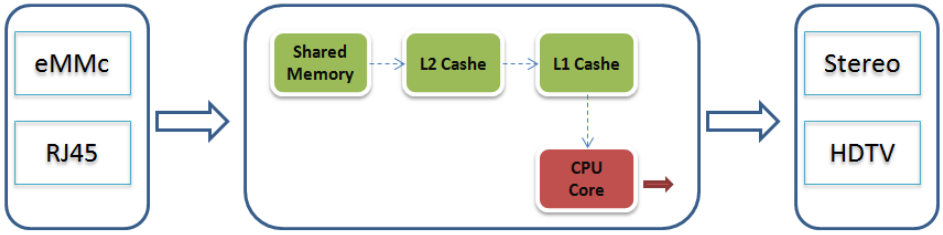

Figure 3.5 BeagleBone Black Considered flow.

All applications are represented in the CSP model as packets following the path in Figure 3.5. All data was captured using ARM DS-5 Development Studio Streamline performance analyzer (www.arm.com/streamline). More details on the model presentation are discussed in Chapter 6.

3.2.2 Platform Architecture For Industrial Case

As industrial platform for our case study we choose BeagleBone Black depicted in Figure 3.6. The BeagleBone is an embedded Linux development board that's a low-cost, community-supported development platform for developers and hobbyists. It's a smaller, more barebones version of the Beagle-board. Both are open source hardware and use Texas Instrument's OMAP processors, which are designed for low-power mobile devices. They boot in Linux under 10 seconds. Get started on development in less than 5 minutes with just a single USB cable.

The BeagleBone board is not only provided with a powerful processor over a typical

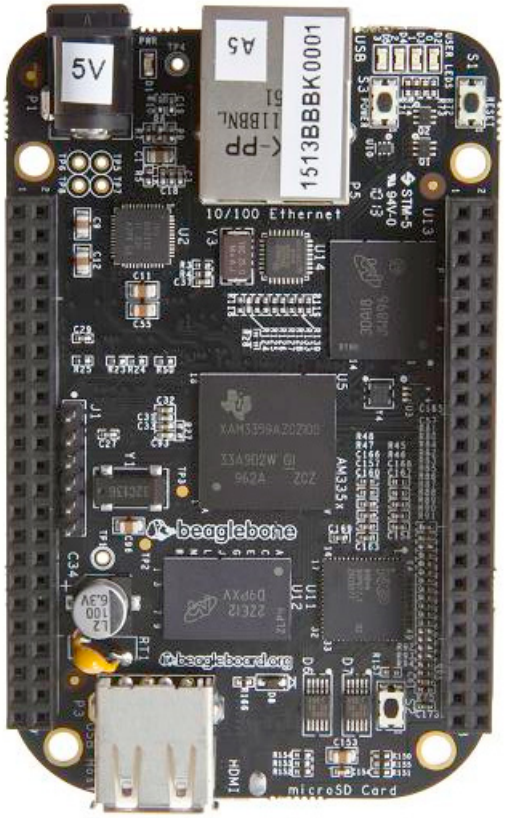

Figure 3.6 BeagleBone Black board ([12]).

micro controller-based board, but it also has some features that make it perfect for immediate development and testing:

- Built-in networking: Not only does the BeagleBone have an on-board Ethernet connection, but all the basic networking tools that come packaged with Linux are available. Several services can be used: like FTP, Telnet, and SSH, or even host your own web server on the board.

- Remote access: Because of its built-in network services, the BeagleBone makes it much easier to access electronics projects remotely over the internet.

- Timekeeping: Without extra hardware, the board can keep track of the date and time of day, and it's updated by pinging internet time servers, ensuring that it's always accurate.

- File system: Just like personal computers, embedded Linux platforms have a built-in file system, so storing, organizing, and retrieving data is a fairly trivial matter.

- Multiple programming languages: C, C++, Python, Perl, Ruby, Java, or even a shell script.

- Multitasking: Unlike a basic microcontroller, embedded Linux platforms can share the processor between concurrently running programs and tasks.

- USB: The BeagleBone can act as both a USB host and a USB device — not only it can be controlled from any computer, also it can be connected to a USB devices. This makes it easy to integrate common USB peripherals like flash drives, wi-fi adapters, and web-cams into any projects.

Figure 3.7 show the high level block diagram of the BeagleBone Black that is considered in our CSP model.

- Processor: Sitara with ARM Cortex-A8 processor offers a proven high-performance solution with millions of units shipped annually. The processor features a high-performance, superscalar microarchitecture with targets ranging from 600MHz to 1GHz and above.

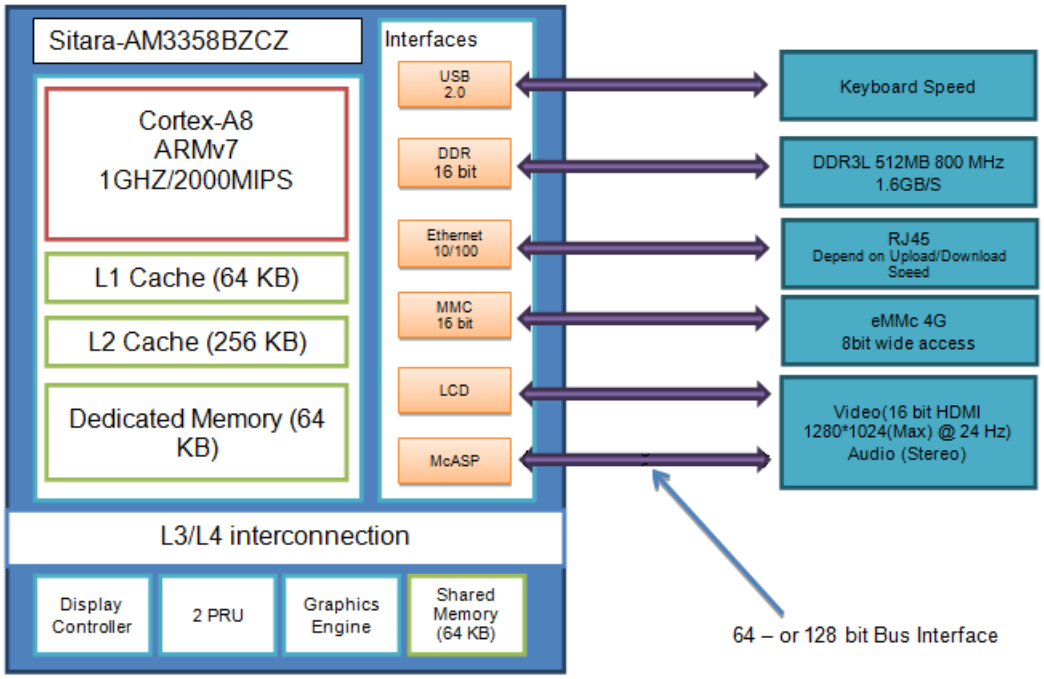

Figure 3.7 BeagleBone Black block digram.

– Memory: There are three memory devices found on the board. Two of them are considered in our CSP model and will be detailed bellow.

  – 512MB DDR3L: A single 256 Mb x 16 DDR3L 4Gb (512MB) memory device is used. The memory used is the MT41K 512 M16HA - 125 from Micron. It will operate at a clock frequency of 303 MHz yielding an effective rate of 606 MHZ on the DDR3L bus allowing for 1.32 GB/S of DDR3L memory bandwidth.

  – 2GB Embedded MMC: A single 2GB embedded MMC (eMMC) device is on the board. The device connects to the MMC1 port of the processor, allowing for 8bit wide access. Default boot mode for the board will be MMC1 with an option to change it to MMC0 for SD card booting. MMC0 cannot be used in 8Bit mode because the lower data pins are located on the pins used by the Ethernet port. This does not interfere with SD card operation but it does make it unsuitable for use as an eMMC port if the 8 bit feature is needed.

– PC USB Interface: The board has a mini USB connector that connects the USB0 port to the processor. This is the same connector as used on the original BeagleBone.

- HDMI Interface: A single HDMI interface is connected to the 16 bit LCD interface on the processor. The 16b interface was used to preserve as many expansion pins as possible to allow for use by the user. The NXP TDA19988BHN is used to convert the LCD interface to HDMI and convert the audio as well. The signals are still connected to the expansion headers to enable the use of LCD expansion boards or access to other functions on the board as needed. The HDMI device does not support HDCP copy protection. Support is provided via EDID to allow the SW to identify the compatible resolutions. Currently the following resolutions are supported via the software:

  - 1280 x 1024
  - 1440 x 900
  - 1024 x 768
  - 1280 x 720

Table 3.1 below shows the main high level features of the Sitara processor. Table 3.2 shows summary on speed/size of BeagleBone components used in the CSP model.

Table 3.1 Sitara Processor Features

Operating Systems	Linux,Android,Windows Embedded CE,QNX, ThreadX	**MMC/SD**	3
Standby Power	7 mW	**CAN**	2
ARM CPU	1 ARM Cortex -A8	**UART (SCI)**	6
ARM MHz (Max.)	275,500,600,800,1000	**ADC**	8-ch 12-bit
ARM MIPS (Max.)	1000,1200,2000	**PWM (Ch)**	3
Graphics Acceleration	1 3D	**eCAP**	3
Other Hardware Acceleration	2 PRU-ICSS,Crypto Accelerator	**eQEP**	3
On-Chip L1 Cache	64 KB (ARM Cortex-A8)	**RTC**	1
On-Chip L2 Cache	256 KB (ARM Cortex-A8)	**I2C**	3
Other On-Chip Memory	128 KB	**McASP**	2
Display Options	LCD	**SPI**	2
General Purpose Memory	1 16-bit (GPMC, NAND flash,NOR Flash,SRAM)	**DMA (Ch)**	64-Ch EDMA
DRAM	1 16-bit (LPDDR-400, DDR2-532, DDR3-606)	**IO Supply (V)**	1.8V(ADC),3.3V
USB Ports	2	**Operating Temperature Range (C)**	-40 to 90

Table 3.2 Summary of BeagleBone components used in the CSP model described in Chapter 6 [12]

Component	**Speed**	**Size**
eMMc	400Mbit/s	4 GB
Network interface	25 Mbit/s (Download speed)	1 task at a time
HDTV	30 fps * frame size(in bits) bit/s Supported frame size: 1280 x 1024 1440 x 900 1024 x 768 1280 x 720	1 task at a time
Stereo	44100 Hz, 2 ch	1 task at a time
Shared memory	1.6 Gbit/s	600 MByte
L2 Cache	1.8 Gbit/s	256 KByte
L1 Cache	2 Gbit/s	256 KByte
CPU core	2 Gbit/s	8 registers

CHAPTER 4 MAPPING PACKET FLOW OF STREAMING APPLICATIONS ONTO MPSOC

The system requirements for an embedded system can be divided into different groups. Each group imposes a number of constraints on the scheduling problem. The functional behaviour of a real-time system is determined by the tasks and resources that constitute the system. Typical functional behaviour requirements are those that control task execution order or task allocation, that is, how a task should execute.

These tasks also have specific behaviour requirements in addition to the functional ones. This behaviour of a task depends mainly on task type, its priority, and how it should interact with another task, that is, when a task should execute. These requirements directly affect the modelling of the application tasks and consequently the construction of the scheduling constraints.

In this chapter we will discuss the construction of the CSP model used with the first platform architecture described in Chapter 3. The input stream is generated in this model as a series of packets. we will look closer at the model constraints and attempt to justify the presence of each constraint construct by examining its origin. We will also discuss how constraints relate to each other, and, based on this information, attempt to identify a minimal set of necessary constraints to be implemented in a scheduling framework.

At the end we will show how we could use this model to identify possible solutions in case of system failure. For example: if the bottleneck is in the limited memory used. This model will allow us to test different possible solutions like increasing the caches or the memory card instead of choosing the cheapest and most convenient one. Also, it helps identifying the possible best performance for the DUT such as trading running time with some of the components capacity and come up with the best combination.

4.1 Constraint-Based Scheduling Approach

The binding of streaming applications onto the target MPSoC architecture is a process with resource limitations and real-time (RT) requirements. Here we use a constraint-based

formulation to model the application-to-architecture mapping, communication routing, flow control, and computation scheduling. We mapped streaming applications onto the target MPSoC architecture by modelling them as Constraint-based scheduling problems. Frames from these applications are translated to sets of packets. Each packet is considered as a task to be done on one of the system resources. The output either gives a suitable schedule for the input stream or it indicates that no solution exists. Our model was implemented using IBM ILOG OPL IDE v6.3 [13] and uses the default search strategy.

4.1.1 Stream model

Among standard types of scheduling problems, our problem is very close to flow shop scheduling, known to be NP-hard. The flow shop scheduling problem consists of a finite set of jobs to be processed on each of a finite set of machines. Jobs have the same processing order through the machines but the order in which uninterruptible jobs are processed on a given machine can vary between machines. Machines generally have a processing capacity and each job-machine pair has its own capacity demand and processing time.

The problem we try to solve follows the same logic as the flow shop scheduling problem: we have a finite set of packets to be processed on a set of system components or resources; each packet-resource pair is called a task; packets have the same processing order through the resources but the order in which non-pre-emptive tasks are processed on a given resource can vary between resources; resources also have a processing capacity and each task has its own capacity demand and processing time. All tasks must be processed before a fixed deadline for the problem. And we try to process it using minimum capacity on each of the system components.

Following the DUT mentioned in Figure 4.1 we have two streaming applications MPEG4 and VOIP (will be referred to as M and V in the equations respectively), each with two sets of frame type (Original $F^m = \{f_i^m\}_{i=1}^{\varphi_m}$ and $F^v = \{f_i^v\}_{i=1}^{\varphi_v}$, Decompressed $F^{m'} = \{f_i^{m'}\}_{i=1}^{\varphi_{m'}}$ and $F^{v'} = \{f_i^{v'}\}_{i=1}^{\varphi_{v'}}$), where $\varphi_m, \varphi_v, \varphi_{m'}, \varphi_{v'}$ represent the number of original and decompressed frames respectively for applications $MPEG4$ and $VOIP$.

Each of the frames is decomposed into a different number of packets depending on the frame and packet size. $f_i^m.size$ and $f_i^v.size$ are the original frame size for applications M and V respectively. As mentioned earlier both sets' values are input from a trace file. $F^{m'}.size$ and $F^{v'}.size$ are the decompressed frame size for applications M and V respectively. These

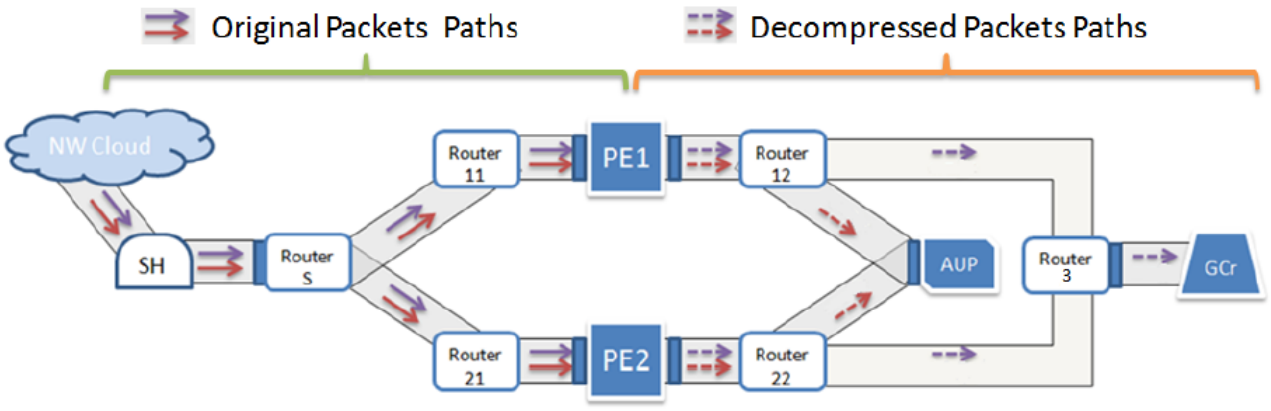

Figure 4.1 Original and decompressed packets in the system floe chart

two values are defined by the type of decompression we use. The packet size through the whole system is fixed and denoted by $P.size$. We denote by $F = F^m \cup F^{m'} \cup F^v \cup F^{v'}$ the set of all frames.

Original packets $P_i^m = \{p_{i,j}^m\}_{j=1}^{\omega_i^m}$ and $P_i^v = \{p_{i,j}^v\}_{j=1}^{\omega_i^v}$ correspond to the streams for frame i received by the DUT into the shared memory and travelling through the system until being processed on one of the two processors. Decompressed packets $P_i^{m'} = \{p_{i,j}^{m'}\}_{j=1}^{\omega_i^{m'}}$ and $P_i^{v'} = \{p_{i,j}^{v'}\}_{j=1}^{\omega_i^{v'}}$ correspond to the streams for frame i generated in the processor as the decompressed packets are the output of the original packets, where $\omega_i^m, \omega_i^v, \omega^{m'}, \omega^{v'}$ represent the number of packets for each frame, calculated as following:

$$\begin{aligned} \omega_i^m &= \lceil f_i^m.size/P.size \rceil, & 1 \le i \le \varphi_m \\ \omega_i^v &= \lceil f_i^v.size/P.size \rceil, & 1 \le i \le \varphi_v \\ \omega^{m'} &= \lceil F^{m'}.size/P.size \rceil \\ \omega^{v'} &= \lceil F^{v'}.size/P.size \rceil \end{aligned}$$

Note that the original frames have a different number of packets depending on the percentage of compression, whereas the number of packets for the decompressed frames is fixed (depending on the display frame size).

Packets travels through the system until they reach the output resource element. We

denote by $P = P^m \cup P^{m'} \cup P^v \cup P^{v'}$ the set of all packets. Note that since the decompressed frame size is always the same, because it depends on the display resolution, the number of decompressed packets for all frames is the same.

Each packet in each stream will be treated as a sequence of tasks, and each task is processed using a single resource. Additional constraints come from the system architecture and applications.

We make the following reasonable assumptions in order to simplify our model:

1. The most critical system resources are buffer capacity and processor frequency, so these will be represented explicitly as resources in our scheduling problem. The NI will be expressed instead by a minimum temporal separation between tasks processed on two consecutive components. Shared memory is not an issue: it is big enough for all packets to stay there as long as they need without violating other constraints.
2. Each packet can be put in one memory location or buffer location.
3. Each memory or buffer location can take only one packet.
4. If a packet is received while the private memory, shared memory, or router buffer is full then it is dropped.
5. The processor speed is the same as the packet transmission speed (bit-rate per second).
6. The shared memory is embedded DRAM with a (single) integrated controller.
7. Packets use Shortest path first in routing decisions.

Then the DUT set of resources is { RS, R_{11}, PE_1, R_{12}, R_{21}, PE_2, R_{22}, R_G, GCr, AuP } (see Chapter 3 for more details) where buffers are represented in routers, processors, the graphics card, and the audio port. Two properties are associated to each resource: speed and capacity.

A packet travels through the system using different resource chains from the receiver (shared memory) to one of the two processors and finally to its output device. The existence of more than one processor makes some of the resources alternatives. For example $\langle RS, R_{11}, PE_1 \rangle$ and $\langle RS, R_{21}, PE_2 \rangle$ are alternative resource chains for original packets before decompression.

4.1.2 Decision Variables

Each packet (e.g. $p^m_{i,j}$) is further decomposed into a set of tasks (e.g. $T^m_{i,j}$) one task per possible resource in the resource chain. We denote by λ_x the number of tasks in $T^x_{i,j}$. Note that in any solution some tasks will not be scheduled since they are associated to alternate resource chains. [1]

Our set of decision variables is then:

$$T = \bigcup_{\substack{1\le i\le\varphi_m \\ 1\le j\le\omega^m_i}} T^m_{i,j} \cup \bigcup_{\substack{1\le i\le\varphi_v \\ 1\le j\le\omega^v_i}} T^v_{i,j} \cup \bigcup_{\substack{1\le i\le\varphi_{m'} \\ 1\le j\le\omega^{m'}}} T^{m'}_{i,j} \cup \bigcup_{\substack{1\le i\le\varphi_{v'} \\ 1\le j\le\omega^{v'}}} T^{v'}_{i,j} \tag{4.1}$$

where

$$T^m_{i,j} = \{t^m_{i,j,k} : 1 \le k \le \lambda_m\}, \qquad T^{m'}_{i,j} = \{t^{m'}_{i,j,k} : 1 \le k \le \lambda_{m'}\}, \tag{4.2}$$

$$T^v_{i,j} = \{t^v_{i,j,k} : 1 \le k \le \lambda_v\}, \qquad T^{v'}_{i,j} = \{t^{v'}_{i,j,k} : 1 \le k \le \lambda_{v'}\}. \tag{4.3}$$

As usual to each task we associate *start*, *end* which are the time the task starts and ends being processed on the corresponding resource respectively, *demand* which is the space it occupies on the component while being processed by it, and *presence* which indicates whether the task is actually scheduled. It is fixed to True for tasks associated with a compulsory resource (i.e. not on an alternative resource chain).

Two last decision variables were used: $StartAfterM$ and $StartAfterV$ which are the time before starting the display for MPEG4 and VOIP applications respectively (once enough frames have been cached) in order to respect their frame rate.

4.1.3 Constraints

Basically we have four main sets of constraints controlling capacity, duration, scheduling start and end time, and dependency. The constraints are chosen to ensure both system and application rules are respected. We do not list all constraints but give a representative of each type of constraint.

1. In ILOG Solver those are handled as "optional" tasks.

Capacity Constraints Given D the simulation deadline, specify the different resources needed by different tasks and the allowed maximum capacity for the constraints.

Constraint 1: Resource capacity must be respected at all time. For example, packets always waits at the current resource until next one free even if they are done (see Figure 4.2). Note that ILOG Solver uses a specialized, optimized constraint named "pulse" to handle such constraints globally [13].

$$\sum_{\substack{1\le i\le\varphi_m\\1\le j\le\omega_i^m}} t^m_{i,j,RS}.demand \le\ RS.Capacity \qquad 0\le d\le D$$

$$such\ that\ t^m_{i,j,RS}.start \le d \le t^m_{i,j,RS}.end$$

$$\sum_{\substack{1\le i\le\varphi_m\\1\le j\le\omega_i^m}} t^m_{i,j,R11}.demand \le\ R11.Capacity \qquad 0\le d\le D$$

$$such\ that\ t^m_{i,j,R11}.start \le d \le t^m_{i,j,R11}.end$$

$$\sum_{\substack{1\le i\le\varphi_m\\1\le j\le\omega_i^m}} t^m_{i,j,PE1}.demand \le\ PE1.Capacity \qquad 0\le d\le D$$

$$such\ that\ t^m_{i,j,PE1}.start \le d \le t^m_{i,j,PE1}.end$$

$$\sum_{\substack{1\le i\le\varphi_m\\1\le j\le\omega_i^m}} t^m_{i,j,R21}.demand \le\ R21.Capacity \qquad 0\le d\le D$$

$$such\ that\ t^m_{i,j,R21}.start \le d \le t^m_{i,j,R21}.end$$

$$\sum_{\substack{1\le i\le\varphi_m\\1\le j\le\omega_i^m}} t^m_{i,j,PE2}.demand \le\ PE2.Capacity \qquad 0\le d\le D$$

$$such\ that\ t^m_{i,j,PE2}.start \le d \le t^m_{i,j,PE2}.end$$

Constraint 2: Alternative tasks must be processed on only one of the alternative resources(see Figure 4.3).

$$t^m_{i,j,R11}.presence \ne t^m_{i,j,R12}.presence, \quad where \quad 1\le i\le\varphi_m \ and \ \ 1\le j\le\omega_i^m$$

$$t^m_{i,j,R11}.presence = t^m_{i,j,PE1}.presence = t^{m'}_{i,j,PE1}.presence = t^{m'}_{i,j,R12}.presence,$$
$$where \quad 1\le i\le\varphi_m \ \ and \ \ 1\le j\le\omega_i^m$$

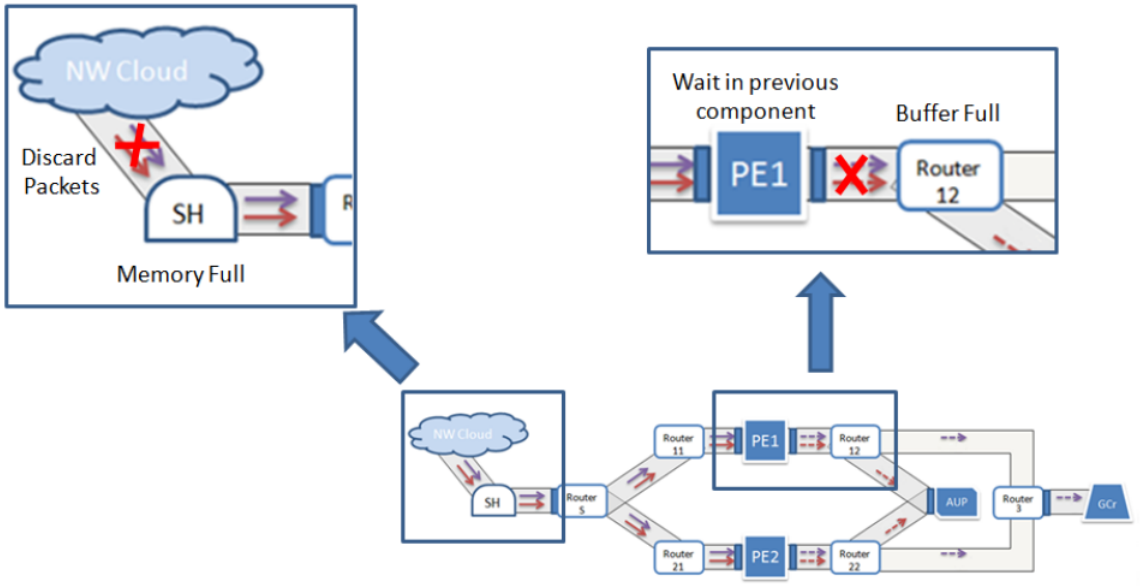

Figure 4.2 Capacity constraints

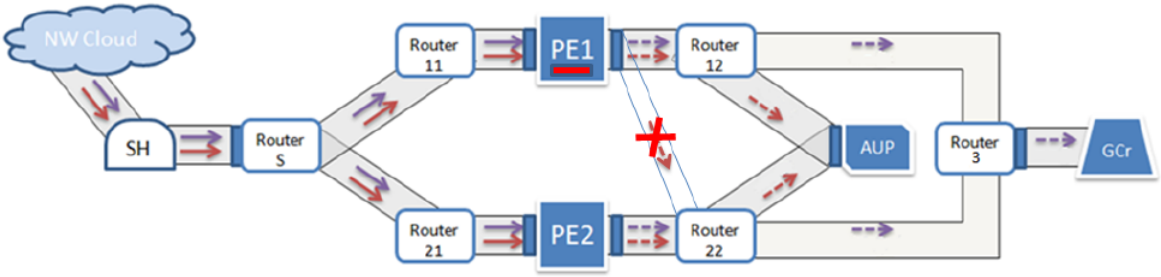

Figure 4.3 Alternative tasks constraints

$$t^m_{i,j,R21}.presence = t^m_{i,j,PE2}.presence = t^{m'}_{i,PE2}.presence = t^{m'}_{i,j,R22}.presence,$$
$$where \quad 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i$$

Duration Constraints Specify the processing duration for different tasks.

Constraint 3: The task duration should be bigger than or equal to the time a certain resource needs to process the application packet according to both resource speed and packet size.

$$t^m_{i,j,RS}.presence = 1 \Rightarrow t^m_{i,j,RS}.duration \geq \tfrac{P.size}{RS.Speed}, \ 1 \leq i \leq \varphi_m \ \ and \quad 1 \leq i \leq \omega^m_i$$

Scheduling start and end time Constraints Specify the processing start and end time for different tasks.

Constraint 4: Given D the simulation deadline, packets must end processing before the simulation deadline. Here we need only to restrict the end on the last resource since packets flow in the system sequentially.

$$t^m_{i,j,\lambda_m}.end \leq D, \ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i$$

Constraint 5: Let $\Upsilon^m_{i,j}$ be the time original packets for application MPEG4 arrive in the system where $1 \leq i \leq \varphi_m and \ \ 1 \leq j \leq \omega^m_i$, SH_{Speed} and NI_{Speed} represent the speed of the shared memory and NI define the minimum temporal separation needed before packets enter RS. Packets must not start processing on the system before their arrival time. Here we need only to restrict the start on the first resource since packets flow in the system sequentially.

$$t^m_{i,j,RS}.start \geqslant \Upsilon^m_{i,j} + \tfrac{P.size}{SH_{Speed}} + \tfrac{P.size}{NI_{Speed}} + 2 * LinkDelay,$$
$$1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i$$

$LinkDelay$ is the delay caused by packet transportation from one resource to the next.

Constraint 6: Given $MaxD$ the maximum time a certain packet can stay on one resource, packets must not stay on a certain resource more then the allowed maximum delay. Here

we have two different types of constraints: the first one to ensure the temporal separation between packets' start time and their appearance on the first resource RS cannot be more than $2MaxD$ since it passes through two components (SH and NI); the second one to restrict $MaxD$ on each resource separately.

$$t^m_{i,j,RS}.start - \Upsilon^m_{i,j} \leq 2MaxD,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i$$

$$t^m_{i,j,k}.duration \leq MaxD,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i \ \ and \ \ 1 \leq k \leq \lambda_m$$

Constraint 7: Decompressed packets of one frame cannot start before completely receiving all packets from their original frame; original packets of one frame cannot end before starting of all packets of their decompressed frames.
Note that we cannot assume the start of decompressed packet must be later than the end of the original ones. Because the original packets stay at the processor longer than their processing time, and at least until the decompressed packets depending on them are processed.

$$t^m_{i,\omega^m_i,\lambda_m}.presence = 1 \Rightarrow t^{m'}_{i,1,1}.start \geq t^m_{i,\omega^m_i,\lambda_m}.start + \tfrac{P.size}{PE.Speed} + linkDelay,\ 1 \leq i \leq \varphi_m$$

$$t^m_{i,\omega^m_i,\lambda_m}.presence = 1 \Rightarrow t^m_{i,\omega^m_i,\lambda_m}.end \geq t^{m'}_{i,\omega^m_i,1}.start + \tfrac{P.size}{PE._{Speed}},\ 1 \leq i \leq \varphi_m$$

Constraint 8: All original frames and there corresponding packets are processed in sequential order on the same component.

$$t^m_{i,j,k}.start \leq t^m_{i,j+1,k}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i - 1 \ \ and \ \ \ 1 \leq k \leq \lambda_m$$
$$t^m_{i,\omega^m_i,k}.start \leq t^m_{i+1,1,k}.start,\ 1 \leq i \leq \varphi_m - 1 \ \ and \ \ \ 1 \leq k \leq \lambda_m$$

Constraint 9: Application MPEG4 uses a periodic pattern known as a Group of Pictures (GOP)[23] that causes a difference in the sequence of data transmitted and data displayed: decompressed frames must follow the order 1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 13, 11, 12. This means that the frames, order will change but the packets order within the same frame will stay the same. We define the corresponding permutation ρ to specify the new GOP frames order (see Figure 4.4).

$$t^m_{i,j,k}.start \leq t^m_{i,j+1,k}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega^m_i - 1 \ \ and \ \ \ 1 \leq k \leq \lambda_m$$

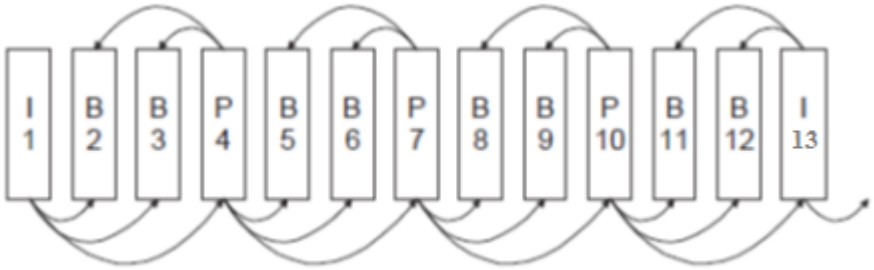

Figure 4.4 MPEG-4 Group Of Picture order and frame dependencies

$$t^{m'}_{\rho_i,\omega_i^{m'},k}.start \leq t^{m'}_{\rho_{i+1},1,k}.start,\ 1 \leq i \leq \varphi_{m'}, 1 \leq k \leq \lambda_{m'}$$

Constraint 10: Given $LinkDelay$ the delay caused by the packet transfer from one resource to the next, tasks for a given packet are processed in order. The end of one task in one resource means its start on the next resource is separated by the $LinkDelay$.

$$t^m_{i,j,RS}.end + LinkDelay = t^m_{i,j,R11}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$
$$t^m_{i,j,R11}.end + LinkDelay = t^m_{i,j,NI11}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$
$$t^m_{i,j,NI11}.end + LinkDelay = t^m_{i,j,PE1}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$
$$t^m_{i,j,RS}.end + LinkDelay = t^m_{i,j,R21}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$
$$t^m_{i,j,R21}.end + LinkDelay = t^m_{i,j,NI21}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$
$$t^m_{i,j,NI21}.end + LinkDelay = t^m_{i,j,PE2}.start,\ 1 \leq i \leq \varphi_m \ \ and \ \ 1 \leq j \leq \omega_i^m$$

Constraint 11: Given τ_i the MPEG frames display time, application MPEG4 decompressed packets of the same frame must respect the video display rate of 30 frames per second.

$$t^{m'}_{i,j,\lambda_{m'}}.end = StartAfterM + \tau_i,\ 1 \leq i \leq \varphi_{m'} \ \ and \ \ 1 \leq j \leq \omega^{m'}$$

Constraint 12: Given UT a constant to ensure application VOIP's decompressed packets of the same frame respect a rate of 50 audio frames per second.

$$t^{v'}_{i,j,\lambda_{v'}}.end = StartAfterV + i * UT,\ 1 \leq i \leq \varphi_{v'} \ \ and \ \ 1 \leq j \leq \omega^{v'}$$

Dependencies Constraints : Some of the tasks depend on others.

Constraint 13: For application MPEG4, frames depending on other frames must be processed on the same processor, (see Figure 4.4).

//IPBB

$t^m_{1,1,R11}.presence = t^m_{i,j,R11}.presence,\ 1 \leq i \leq 4\ \ and\ \ 2 \leq j \leq \omega^m_i$

$t^m_{1,1,R21}.presence = t^m_{i,j,R21}.presence,\ 1 \leq i \leq 4\ \ and\ \ 2 \leq j \leq \omega^m_i$

//PBB,IBB

$t^m_{i,1,R11}.presence = t^m_{l,j,R11}.presence,\ 5 \leq i \leq \varphi_m - 1\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 1$

$t^m_{i,1,R21}.presence = t^m_{l,j,R21}.presence,\ 5 \leq i \leq \varphi_m - 1\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 1$

$t^m_{i,1,R11}.presence = t^m_{l,j,R11}.presence,\ 5 \leq i \leq \varphi_m - 2\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 2$

$t^m_{\varphi_m,1,R11}.presence = t^m_{\varphi_m,j,R11}.presence, 1 \leq j \leq \omega^m_i\ \ and\ \ \varphi_m\ mod\ 3 = 2$

$t^m_{i,1,R21}.presence = t^m_{l,j,R21}.presence,\ 5 \leq i \leq \varphi_m - 2\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 2$

$t^m_{\varphi_m,1,R21}.presence = t^m_{\varphi_m,j,R21}.presence, 1 \leq j \leq \omega^m_i\ \ and\ \ \varphi_m\ mod\ 3 = 2$

$t^m_{i,1,R11}.presence = t^m_{l,j,R11}.presence,\ 5 \leq i \leq \varphi_m - 3\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 0$

$t^m_{\varphi_m-1,1,R11}.presence = t^m_{i,j,R11}.presence,\ varphi_m{-}1 \leq i \leq \varphi_m\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ \varphi_m\ mod\ 3 = 0$

$t^m_{i,1,R21}.presence = t^m_{l,j,R21}.presence,\ 5 \leq i \leq \varphi_m - 3\ \ and\ \ i \leq l \leq i+3\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ i\ mod\ 3 = 2\ \ and\ \ \varphi_m\ mod\ 3 = 0$

$t^m_{\varphi_m-1,1,R21}.presence = t^m_{i,j,R21}.presence,\ varphi_m{-}1 \leq i \leq \varphi_m\ \ and\ \ 1 \leq j \leq \omega^m_i\ \ and\ \ \varphi_m\ mod\ 3 = 0$

$t^m_{i,RS+1}.presence = t^m_{j,RS+1}.presence,\ 5 \leq i \leq \varphi_m - 1,\ i \leq j \leq i+3$

Constraint 14: For application VOIP, packets of the same frame must be processed on the same processor.

$t^v_{1,1,R11}.presence = t^v_{i,j,R11}.presence,\ 1 \leq i \leq \varphi_v\ \ and\ \ 2 \leq j \leq \omega^v_i$

A detailed discussion for experiments and results will follow in next section.

[htbp!]

4.2 Experimental Results

Our aim is to experimentally identify some non-trivial cases of system failure which are unlikely to be detected manually by a Test Engineer. Particular interest is given to cases that show the impact of independent applications sharing the same computational resources. Our model was implemented using IBM ILOG OPL IDE v6.3 and used the default search [39].

The CSP model created simulating one second running simulation is summarized as:

! ————————————————————————————————————-

! Satisfiability problem - 25,136 variables, 89,763 constraints

Table 4.1 Design space for the experimental platform

Parameter	From	To
PE BW	1Mb/s	512Mb/s
PE Memory Size	1.5KB	15KB
Bus Latency	10ns	100ns
Max Delay	2ms	1s
Frame size	QCIF	SDTV
Simulation deadline	2 seconds	

Table 4.2 experimental Results: the 2^{nd} line indicate PE size, 3^{rd} line Determine application specifications, and 4^{th} line is Bus Delay. note the results with f symbol indicate that solution found in less then 15 second

	Application M					Application V					Applications $M+V$								
	$\geq 6K$				$<6K$	$\geq 3K$				$<3K$	$\geq 6K$								$<6K$
	$(QCIF)$		$(SDTV)$			$(2ms)$		$(1s)$			$(QCIF,2ms)$		$(QCIF,1s)$		$(SDTV,2ms)$		$(SDTV,1s)$		
PE BW	10ns	100ns	10ns	100ns		10ns	100ns	10ns	100ns		10ns	100ns	10ns	100ns	10ns	100ns	10ns	100ns	
1	f	f	f	f	f	f	f	–	–	–	f	f	f	f	f	f	f	f	f
2	f	f	f	f	f	f	f	–	–	–	f	f	f	f	f	f	f	f	f
4	f	f	f	f	f	f	f	–	–	–	f	f	f	f	f	f	f	f	f
8	–	–	f	f	f	f	f	1	1	f	f	f	–	–	f	f	f	f	f
11	69	65	f	f	f	f	f	1	1	f	f	f	f	f	f	f	f	f	f
16	52	73	f	f	f	f	f	1	1	f	f	f	76	66	f	f	f	f	f
32	68	82	f	f	f	f	f	1	1	f	f	f	71	72	f	f	f	f	f
64	93	86	f	f	f	2	2	1	1	f	388	386	97	102	f	f	f	f	f
128	98	85	–	–	–	2	2	1	1	f	379	418	168	154	–	–	–	–	–
256	91	88	–	–	–	2	2	1	1	f	406	432	279	275	–	–	–	–	–
512	103	87	–	–	–	2	3	1	1	f	431	510	276	281	–	–	–	–	–

! Presolve : 24,409 extractables eliminated
! FailLimit = 10
! RandomSeed = 2
! RestartFailLimit = 50
! Initial process time : 17.79s (17.54s extraction + 0.25s propagation)
! . Log search space : 592,706.4 (before), 592,706.4 (after)
! . Memory usage : 198.1 MB (before), 204.3 MB (after)
! Using parallel search with 8 workers.
! ————————————————————————————————————-

All experiments were run on Intel Core i7 computer with 4GB RAM. The target configuration parameters are shown in Table 4.1.

We take our results shown in Table 4.2 for M (with two different frame sizes) and V (with either a delay restriction like a phone call, or some allowed buffering flexibility like voice message) separately and then combined. Entries labelled "f" indicate a proven system failure, i.e. the solver showed that there is no solution. Entries labelled "–" indicate that the solver could neither find a feasible schedule nor prove that there is none.

Our tests show how the applications affect each other. For example, when M is dealing with an SDTV stream with a PE BW of 64Mb/s, the non-restricted delay version of V will always fail. Similarly, the delay-restricted version of V will always fail when M is processing a QCIF stream with a PE BW of 64Mb/s

Our methodology has also allowed us to identify some corner cases, such as the combination of M and V will fail with a PE BW 11 Mb/s, even if both applications can be successfully scheduled independently, meanintg that methodology can identify issues due to the non-obvious interaction of multiple applications.

Assuming that PEs need to be fast enough to produce the required frame rate, one can devise the following empirical rule:

$$\frac{F_s fps/P_s}{Out_{max}} > 1 \tag{4.4}$$

with F_s, P_s the frame and packet size, respectively, and Out_{max} the maximum processor output.

When applying this rule, the DUT should fail when running only M using QCIF with a PE BW $<$ 14.5 Mb/s. Nevertheless, our methodology shows that a PE BW = 11 Mb/s is sufficient for the application. This shows that non-trivial optimizations can be discovered with our methodology.

Conversely for V, a PE BW = 6.8 Kb/s should be sufficient, but we can observe that in delay-restricted conditions the system could not be scheduled if PE BW $<$ 64 Mb/s. This shows we can detect issues related to buffering and link delays.

To test the performance of our methodology, we compared the it with the use of the ReSP MPSoC Simulation Platform [6]. When using FFMPEG, our system can detect system failure in less then 15 second and verify system success in 10 minutes, as opposed to ReSP, which takes around 30 minutes to run a single simulation.

CHAPTER 5 MAPPING FRAME FLOW OF STREAMING APPLICATIONS ONTO MPSOC

5.1 Alternative Model

Although the previous model in Chapter 4 did define some corner cases, unfortunately it will not scale very well because of its large number of tasks (typically hundreds of thousands). This is why we created a new model directly from frames tasks, which will significantly reduce the number of tasks needed to be scheduled (a few hundreds) and at the same time will not violate the system constraints as we will take into consideration new constraints to ensure system validity.

The scheduling problem will use the same set of frames, the same set of resources and, the same processing chains used before except that we will add NI as one of the resources to ensure a more correct timing between different resources as the problem becomes more complex.

5.1.1 Stream model

The model will have the same two streaming applications MPEG4 and VOIP (will be referred to as M and V in the equations respectively), each with two sets of frame type (Original $F^m = \{f_i^m\}_{i=1}^{\varphi_m}$ and $F^v = \{f_i^v\}_{i=1}^{\varphi_v}$, Decompressed $F^{m'} = \{f_i^{m'}\}_{i=1}^{\varphi_{m'}}$ and $F^{v'} = \{f_i^{v'}\}_{i=1}^{\varphi_{v'}}$), where $\varphi_m, \varphi_v, \varphi_{m'}, \varphi_{v'}$ represent the number of original and decompressed frames respectively for applications $MPEG4$ and $VOIP$. Note that for our case study since we are dealing with frames and not packets, the number of original frames is the same as the number of decompressed frames: they differ only in size (i.e. $\varphi_m = \varphi_{m'}$ and $\varphi_v = \varphi_{v'}$), also the packet size through the whole system is fixed and denoted by P.size.

Then the DUT set of resources is { RS, R_{11}, NI_{11}, PE_1, NI_{12}, R_{12}, R_{21}, NI_{21}, PE_2, NI_{22}, R_{22}, R_G, NI_G, GCr, NI_A, AuP } (see Chapter 3 for more details)

5.1.2 Decision Variables

Each frame (e.g. f_i^m) is further decomposed into a set of tasks (e.g. T_i^m) one task per possible resource in the resource chain. We denote by λ_x the number of tasks in T_i^x. Note that in any solution some tasks will not be scheduled since they are associated to alternate resource chains.

Our set of decision variables is then:

$$T = \bigcup_{1\leq i\leq \varphi_m} T_i^m \cup \bigcup_{1\leq i\leq \varphi_v} T_i^{m'} \cup \bigcup_{1\leq i\leq \varphi_{m'}} T_i^v \cup \bigcup_{1\leq i\leq \varphi_{v'}} T_i^{v'} \quad (5.1)$$

where

$$T_i^m = \{t_{i,k}^m : 1 \leq k \leq \lambda_m\}, \qquad T_i^{m'} = \{t_{i,k}^{m'} : 1 \leq k \leq \lambda_{m'}\}, \quad (5.2)$$

$$T_i^v = \{t_{i,k}^v : 1 \leq k \leq \lambda_v\}, \qquad T_i^{v'} = \{t_{i,k}^{v'} : 1 \leq k \leq \lambda_{v'}\}. \quad (5.3)$$

As usual to each task we associate $start, end$ which are the time the task starts and ends being processed on the corresponding resource respectively, *demand* which is the space it occupies on the component while being processed by it, and *presence* which indicates whether the task is actually scheduled.
It is fixed to True for tasks associated with a compulsory resource (i.e. not on an alternative resource chain).

Two last decision variables were used: $StartAfterM$ and $StartAfterV$ which are the time before starting display for MPEG4 and VOIP applications respectively (once enough frames have been cached) in order to respect their frame rate.

5.1.3 Constraints

Basically we have four main sets of constraints controlling capacity, duration, scheduling start and end time, and dependency. The constraints are chosen to ensure both system and application rules are respected. We do not list all constraints but give a representative of each type of constraint.

Capacity Constraints Given D the simulation deadline, specify the different resources needed by different tasks and the allowed maximum capacity for the constraints.

Constraint 1: Resource capacity must be respected at all time. Note that ILOG solver use a specialized, optimized constraint named "pulse" to handle such constraints globally [13].

$$\sum_{1 \le i \le \varphi_m} t^m_{i,RS}.demand \le \ RS.Capacity, \quad 0 \le t \le D$$
$$such\ that\ t^m_{i,RS}.start \le t \le t^m_{i,RS}.end$$

Constraint 2: Alternative tasks must be processed on only one of the alternative resources.

$t^m_{i,R11}.presence \neq t^m_{i,R12}.presence, \quad 1 \le i \le \varphi_m$

$$t^m_{i,R11}.presence = t^m_{i,PE1}.presence = t^{m'}_{i,PE1}.presence = t^{m'}_{i,R12}.presence,$$
$$1 \le i \le \varphi_m$$
$$t^m_{i,R21}.presence = t^m_{i,PE2}.presence = t^{m'}_{i,PE2}.presence = t^{m'}_{i,R22}.presence,$$
$$1 \le i \le \varphi_m$$

Duration Constraints Specify the processing duration for different tasks.

Constraint 3: As the task duration is not only the time a certain resource needs to process it but also we need to consider the delays added if this was sent as packets and also that any delay on a previous resource must affect the duration on the next one. Here we need to know the number of packets on each frame:

$p^m_i = \lceil \frac{f^m_i.size}{P.size} \rceil, \ 1 \le i \le \varphi_m$
$p^v_i = \lceil \frac{f^v_i.size}{P.size} \rceil, \ 1 \le i \le \varphi_v$
$p^{m'} = \lceil \frac{f^{m'}.size}{P.size} \rceil$
$p^{v'} = \lceil \frac{f^{v'}.size}{P.size} \rceil$

Note that the original packets have a different number of frames depending on the percentage of compression, whereas the number of packets for the decompressed frames is fixed (to the display frame size). Let Υ^m_i be the time original packets for application MPEG4

arrive in the system where $1 \leq i \leq \varphi_m$, SH_{Speed} and NI_{Speed} represent the shared memory and NI speed defined the minimum temporal separation needed before packets enter RS. Because the first task cannot consider a delay from the previous components its duration is calculated differently:

$$t^m_{i,RS}.duration \geq \frac{f^m_i.size}{RS.Speed} + linkDelay * p^m_i + t^m_{i,RS}.start \\ -(\Upsilon^m_i + \frac{f^m_i.size}{SH_{Speed}} + \frac{f^m_i.size}{NI_{Speed}}),\ 1 \leq i \leq \varphi_m$$

All other components follow the same rule so here we show only the duration for the second task.

$$t^m_{i,2}.precence = 1 \Rightarrow t^m_{i,2}.duration \geq \frac{f^m_i.size}{RS.Speed} + linkDelay * p^m_i + t^m_{i-1,RS}.duration \\ -(\frac{f^m_{i-1}.size}{RS.Speed} + linkDelay * p^m_{i-1}),\ 1 \leq i \leq \varphi_m$$

Scheduling start and end time Constraints Specify the processing start and end time for different tasks.

Constraint 4: Given D the simulation deadline, frames must end processing before the simulation deadline.

$$t^m_{i,\lambda_m}.end \leq D,\ 1 \leq i \leq \varphi_m$$

Constraint 5: Packets must not start processing on the system before their arrival time. Here we need only to restrict the start on the first resource since packets flow in the system sequentially.

$$t^m_{i,RS}.start \geqslant \Upsilon^m_i + \frac{f^m_i.size}{SH_{Speed}} + \frac{f^m_i.size}{NI_{Speed}} + 2 * linkDelay * p^m_i, \\ 1 \leq i \leq \varphi_m$$

Constraint 6: Given $MaxD$ the maximum time a certain frame can stay on one resource, frames must not stay on a certain resource more then the allowed maximum delay. Here we have two different types of constraints: the first one to ensure the temporal separation between frames' start time and their appearance on the first resource RS cannot be

more than $2MaxD$ since it passes through two components (SH and NI); the second one to restrict $MaxD$ on each resource separately.

$$t^m_{i,RS}.start - \Upsilon^m_i \leq 2MaxD,\ 1 \leq i \leq \varphi_m$$

$$t^m_{i,k}.duration \leq MaxD,\ 1 \leq i \leq \varphi_m,\ 1 \leq k \leq \lambda_m$$

Constraint 7: Decompressed frames cannot start before completely receiving their original frame; original frames cannot end before starting their decompressed frames.

$$t^{m'}_{i,1}.start \geq t^m_{i,\lambda_m}.start + \frac{f^m_i.size}{PE.Speed} + linkDelay * p^m_i + t^m_{i-1,1}.duration - (\frac{f^m_{i-1}.size}{NI.Speed} + linkDelay * p^m_{i-1}),\ 1 \leq i \leq \varphi_m$$

$$t^m_{i,\lambda_m}.end \geq t^{m'}_{i,1}.start + \frac{P.size}{PE.Speed},\ 1 \leq i \leq \varphi_m$$

Constraint 8: All original frames are processed in sequential order on the same component.

$$t^m_{i,k}.start \leq t^m_{i+1,k}.start,\ 1 \leq i \leq \varphi_m - 1,\ 1 \leq k \leq \lambda_m$$

Constraint 9: Application MPEG4 uses a periodic pattern known as a Group of Pictures (GOP)[23] that causes a difference in the sequence of data transmitted and data displayed: decompressed Frames must follow the order 1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 13, 11, 12. We define the corresponding permutation ρ to specify the new GOP frames order (see Figure 5.1).

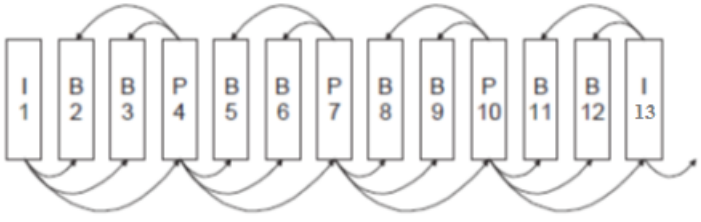

Figure 5.1 MPEG-4 Group Of Picture order and frame dependencies

$$t^{m'}_{\rho_i,k}.start \leq t^{m'}_{\rho_{i+1},k}.start,\ 1 \leq i \leq \varphi_m, 1 \leq k \leq \lambda_{m'}$$

Constraint 10: Given *LinkDelay* the delay caused by the frame transportation from one resource to the next, tasks for a given packet are processed in order. Here we allow tasks of one frame to start on the next resource as soon as one packet has been processed.

$$t^m_{i,RS}.start + LinkDelay + \frac{P.size}{RS.Speed} \leq t^m_{i,R11}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,R11}.start + LinkDelay + \frac{P.size}{R11.Speed} \leq t^m_{i,NI11}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,NI11}.start + LinkDelay + \frac{P.size}{NI11.Speed} \leq t^m_{i,PE1}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,RS}.start + LinkDelay + \frac{P.size}{RS.Speed} \leq t^m_{i,R21}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,R21}.start + LinkDelay + \frac{P.size}{R21.Speed} \leq t^m_{i,NI21}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,NI21}.start + LinkDelay + \frac{P.size}{NI21.Speed} \leq t^m_{i,PE2}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,k}.end - t^m_{i,k+1}.start \leq t^m_{i,k}.duration + linkDelay,\ 1 \leq i \leq \varphi_m,\ 1 \leq k \leq \lambda_m$$

Constraint 11: Given τ_i the MPEG frames display time, application MPEG4 decompressed frames must respect the video display rate of 30 frames per second.

$$t^{m'}_{i,\lambda_{m'}}.end = StartAfterM + \tau_i,\ 1 \leq i \leq \varphi_{m'}$$

Constraint 12: Given UT a constant to ensure application VOIP's decompressed packets of the same frame respect a rate of 50 audio frames per second.

$$t^{v'}_{i,\hat{t}^{v'}}.end = StartAfterV + i * UT,\ 1 \leq i \leq \varphi_{v'}$$

Dependencies Constraints : Some of the tasks depend on others.

Constraint 13: For application MPEG4, frames depending on other frames must be processed on the same processor, (see Figure 5.1). Constraints are shown for complete IPBB, PBB, IBB sequences for simplicity but all cases are considered in the model.

//IPBB
$t^m_{1,RS+1}.presence = t^m_{i,RS+1}.presence,\ 1 \leq i \leq 4$
//PBB,IBB

$t^m_{i,RS+1}.presence = t^m_{j,RS+1}.presence,\ 5 \leq i \leq \varphi_m - 1,\ i \leq j \leq i + 3$

A detailed discussion for experiments and results will be followed in next section.

5.2 Experimental Results

Our aim is to experimentally identify non-trivial cases when the system fails to meet its performance goals, in particular if they are unlikely to be detected manually by a test engineer. We also aim at finding corner cases where there is a chance of succesfully meeting the performance goals, and it is worth investing time for testing in higher detail. Particular interest is given to cases that show the impact of competition between independent applications sharing the same computational resources. All experiments were run on an Intel Core i7 computer with 8GB RAM. The target parameters are shown in Table 5.1. The design space is explored by manually and sequentially applying these parameters. Because In the synchronous dataflow (SDF) MoC, the static data rate allows for the construction of periodic schedules with bounded memory size at compile time [43], simulation deadline defined consider enough to ensure the validity of infinite behaviour of the system.

The CSP model created to represent one second summarized as follow:

```
! ---------------------------------------------------------
! Satisfiability problem - 11,903 variables, 58,999 constraints
! Presolve : 26,658 extractables eliminated
! SearchType = MultiPoint
! SolutionLimit = 1
! Initial process time : 2.20s (2.08s extraction + 0.11s propagation)
! . Log search space : 148,970.0 (before), 148,970.0 (after)
! . Memory usage : 104.0 MB (before), 112.1 MB (after)
! Using parallel search with 8 workers.
! ---------------------------------------------------------
```

We tested our model with 50 different system configurations. The results came to confirm the one published on [16] with more gain offered by the new model.

Results are shown in Figure 5.2 and Figure 5.3 for different processing element bandwidths. These are 2^n where $n = 0, 1, 2, ..., 11$ (on the vertical axis), for different architecture

Table 5.1 Design space for the experimental platform

Parameter	From	To	Parameter	Value
Processing Element (PE) Size	1.5KB	15KB	Packet Size	1.5 KB
PE Band Width (BW)	1Mb/s	512Mb/s	Num. of PEs	2
Bus Latency	10ns	100ns	Memory BW	2 Gb
Max Allowed VOIP Delay	2ms	1s	Buffer Size	1.5 KB
MPEG frames Size	$QCIF$	$SDTV$	Simulation deadline	2 seconds

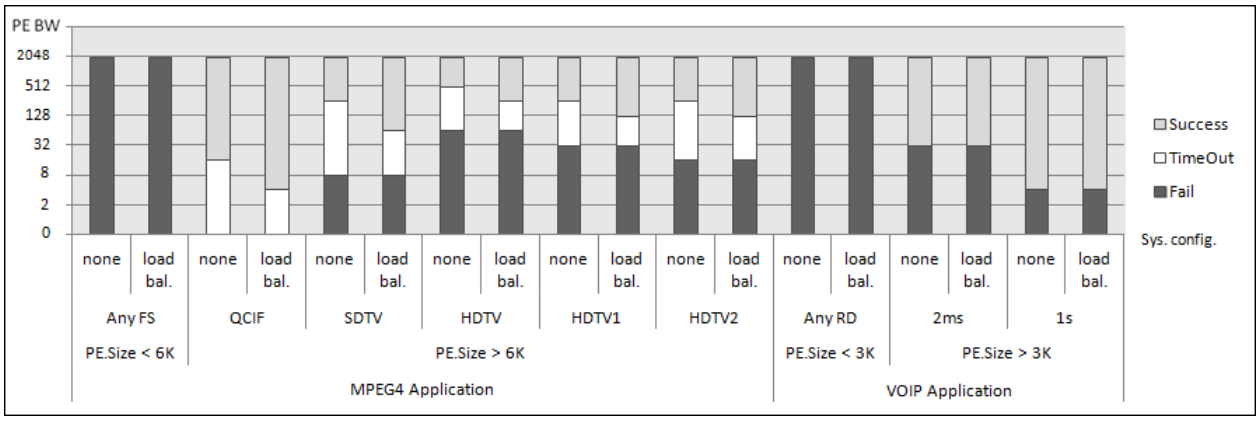

Figure 5.2 Results for MPEG4 and VOIP separately

configurations (on the horizontal axis). We used five different frame sizes for $MPEG4$ (with FS meaning that results apply to all frame sizes), and two configurations for $VOIP$ (delay-restricted and non-restricted, with RD meaning that results apply to both versions). The two applications were tested separately, and then combined to study the impact of one application on the other. The last parameter we considered is the scheduling policy associated to the processors: either load balancing or FIFO.

We used three different terms to represent the results for each architecture: “success” indicates that a solution was found at this level of detail, and it can be investigated further; “failure” indicates that the solver showed that there is no solution; “timeout”, indicates that the solver could neither find a feasible schedule nor prove that there is none for this bandwidth within a 10-minutes time limit.

Figure 5.2 shows results for the model (”none”, ”load bal.”) where ”none” indicates that no processor scheduling policy was used, and ”load bal.” indicates that load balance processor scheduling policy was used, running only one application and Figure 5.3 show results when combining the two applications.

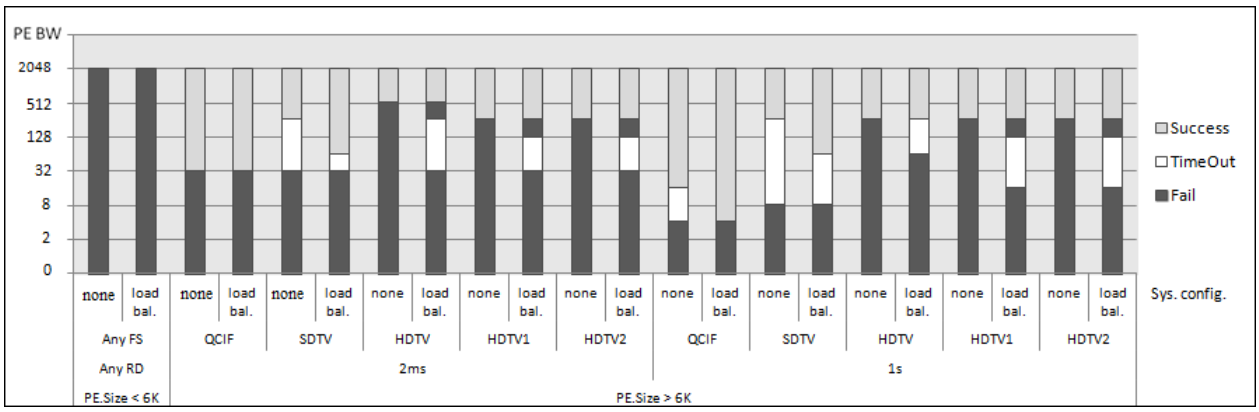

Figure 5.3 Results for MPEG4 and VOIP combined

Adding load balancing generally gives better results, except in some cases shown in Figure 5.3 where the solver could not prove the absence of solutions when the two applications are combined. This does not reduce the value of the model since the results prove the lack of solution with any processor scheduling policy.

Some tests gave straightforward results: both applications will always fail if the private memory of the PEs is less than 3 KB and 6 KB for MPEG4 and VOIP, respectively. This is due to inter- and intra-frame packet dependencies.

Other tests show how one application failure might affect the other, when running simultaneously. For example, when $MPEG4$ is dealing with an SDTV stream with a PE BW of 64Mb/s, the non-restricted delay version of $VOIP$ will always fail. Similarly, the delay-restricted version of $VOIP$ will always fail when MPEG4 is processing a QCIF stream with a PE BW of 64Mb/s.

But our methodology also allowed us to identify some interesting cases. For example the combination of $MPEG4$ and $VOIP$ will fail with a PE BW 11 Mb/s even if both applications can be successfully scheduled independently, meaning that our methodology can identify issues due to the non-obvious interaction of multiple applications. Please note that this case is not shown in the figure; we narrowed in on this particular critical point.

Given that PEs need to be fast enough to produce the required frame rate, one can devise the following empirical rule: $\frac{F_s fps/P_s}{Out_{max}} > 1$ with F_s, P_s the frame and packet size, respectively, and Out_{max} the maximum processor output. Hence the DUT should fail when running only $MPEG4$ using QCIF with a PE BW $<$ 14.5 Mb/s. Nevertheless our methodology shows that a PE BW = 11 Mb/s is sufficient for the application. This shows that non-trivial optimizations can be discovered as well with our methodology. Conversely, for $VOIP$, a PE BW = 6.8 Kb/s should be sufficient, but we can observe that in delay-restricted conditions the system could not be scheduled with PE BW $<$ 64 Mb/s. This shows we can detect issues related to buffering and link delays.

By analyzing our results, we can recommend an architecture to run these two applications using requirements such as a minimum processor private memory of 6KB, and a reasonable processor bandwidth of 1 Gb/s to run the more demanding application.

To test the performance of our methodology, we compared it with the use of the ReSP MPSoC Simulation Platform [6]. When using FFMPEG, our system can detect system failure in less than 15 seconds and verify system success in 10 minutes, while ReSP takes around 30 minutes to run a single simulation.

CHAPTER 6 MAPPING TASKS FLOW OF STREAMING APPLICATIONS ONTO MPSOC

The next step in our work is to test the proposed methodology on an already existing architecture by running selected applications and by comparing results with our models.

6.1 Industrial-Case Model

As mentioned in Chapter 3, BeagleBone Black (BBB) is our test architecture platform. Five applications were used to represent interactive computations on the BBB to measure system performance:

- (Dhrystone—WhestStone) from UnixBench benchmark: the main purpose of running these is to generate activity on the cache memory and processor. This will help discover critical cases when interacting with other applications.
- MPlayer movie player: MPlayer is a source of test cases that need synchronization between different tasks (audio/video), it must respect a certain task order (the one in which frames are received), and has a minimum performance target (the minimum display rate is 30 frame/sec). The application was used in two different modes: real-time stream and local stream. Both modes were tested with different video rates, resolution, and quality in order to create diverse test cases with different system needs and constraints.
- (Iceweasle—Wget): both applications are used to create more activity on the cache memory, processor, and network interface. This will help create more critical test cases when interacting with other applications.

Taking the BBB with the five previously mentioned application, we reused the model described in Chapter 5 to create a constraint-based scheduling problem for our industrial case and generate the appropriate test cases for it. A detailed discussion is provided through the following sections.

6.1.1 Stream model

Each application has a number of instructions that must be executed through different system components, which in turn causes a number of memory access hits and misses through the system. We use these numbers to calculate applications activity requirements and performance throughout the system. The number of instructions/hits-misses are huge (hundreds of

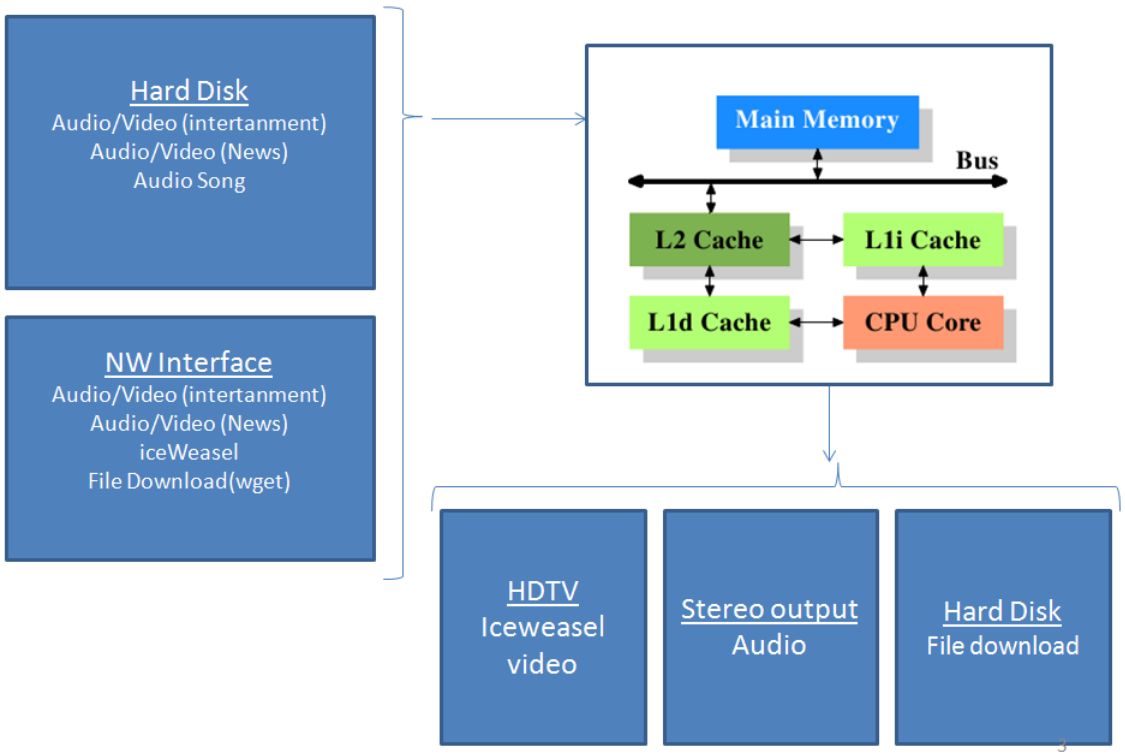

Figure 6.1 streaming applications flow in MPSoC industrial architecture.

millions of instructions) and require a very small fraction of time to be done(being read and copied from one resource to the next along with data input/output needed to be correctly executed). This makes it impossible to be expressed as a CSP scheduling problem. This is due to the fact that Large-Scale scheduling problems are NP-hard [22]. Accordingly, in our model we will assume that each application has one job to be run on each of the components in its running path. The number of tasks needed by each of the jobs depends on the maximum capacity of the component and the minimum time fraction that can be considered in the model. We explain it further in the next few paragraphs.

Following the DUT in Figure 6.1, our model was applied to 8 different streaming applications. Application 1 and 2 represent MPlayer application video and audio streams running from the local mass memory (will be referred to as Al and Vl in the equations respectively). Application 3 and 4 represent the MPlayer application video and audio streams running as a live stream from an online website (will be referred to as An and Vn in the equations, respectively). Application 5 represent downloading files from the Internet using the Wget tool (will be referred to as Wg in the equations). Application 6 represent the ice-weasel web-browser opening a different sequence of websites (will be referred to as Iw in the equations). Applications 7 and 8 are both benchmarks (Dhrystone—WhestStone), used to stress cache

memory and processor (will be referred to as Dh and Wh in the equations respectively) – See Table 6.1 for more details.

Each of the applications from 1 to 8 is represented by a set of jobs: $J^{al} = \{j_i^{al}\}_{i=1}^{\varphi_{al}}$, $J^{vl} = \{j_i^{vl}\}_{i=1}^{\varphi_{vl}}$, $J^{an} = \{j_i^{an}\}_{i=1}^{\varphi_{an}}$, $J^{vn} = \{j_i^{vn}\}_{i=1}^{\varphi_{vn}}$, $J^{wg} = \{j_i^{wg}\}_{i=1}^{\varphi_{wg}}$, $J^{iw} = \{j_i^{iw}\}_{i=1}^{\varphi_{iw}}$, $J^{dh} = \{j_i^{dh}\}_{i=1}^{\varphi_{dh}}$, $J^{wh} = \{j_i^{wh}\}_{i=1}^{\varphi_{wh}}$, where φ_{al}, φ_{vl}, φ_{an}, φ_{vn}, φ_{wg}, φ_{iw}, φ_{dh}, φ_{wh} represent the number of jobs for each of the applications Al, Vl, An, Vn, Wg, Iw, Dh, and Wh, respectively. Note that for our case study, since each job is done in only one system component, the number of jobs and their sequences are the same as the path each application has to go through in the system in order to be complete.

The DUT set of resources is $\{RJ, HD, SM, PE, LC_1, LC_2, GCr, AuP\}$ (see Chapter 3 for more details). The application requirements on each of these resources will be considered as the amount of traffic generated inside this resource or component in order to finish all applications, tasks and successfully run on the system.

As mentioned earlier, we are using the BBB which has an ARM Cortex-A8 as its processor core. The Cortex-A8 uses c9, the Event SELection (EVTSEL) Register, to select the events that you want a Performance Monitor Count Register to count.

The EVTSEL Register is:

– a read/write register common to Secure and Insecure states

– accessible as determined by c9, User Enable Register.

Different application activities on each of the BBB system components are measured using the DS-5 streamline performance analyzers. The main problem in estimating the minimum requirement for each application running through the system is to combine the traffic generated by each of the applications inside the processor core. We use 5 counters (Figure 6.2) to calculate this information:

- **Data Access:** memory read and write operations that cause a cache access to at least the level of data or unified cache closest to the processor;
- **L2 Access (0x43):** any access to L2 cache;
- **L2 miss (0x44):** any cache-able miss on L2 cache;
- **Instructions Executed (0x08):** Instructions architecturally executed. This counter counts for all instructions, including conditional instructions that fail their condition codes;
- **L1 inst Hash miss (0x4a):** any L1 instruction memory access that misses in the cache as a result of the hashing algorithm. The cases covered are:
  - Hash hit and physical address miss;
  - Hash hit and physical address hit in another way;

Table 6.1 Details of Streaming Application Running on the Industrial Platform

Application name	Application case description
MPlayer local audio play (Al)	This application will be running in two different cases, one is as a stand alone application representing a song playing from a local storage device. The other case is when this application is synchronized with MPlayer local video play to represent audio/video movie running from a local storage device.
MPlayer local video play (Vl)	This application will be running in two different cases, one is as a stand alone application representing a video playing from a local storage device. The other case is when this application is synchronized with MPlayer local audio play to represent audio/video movie running from a local storage device.
MPlayer live audio play (An)	This application will be running in two different cases, one is as a stand alone application representing a live stream playing song. The other case is when. This application is synchronized with MPlayer live video stream to represent audio/video movie play
MPlayer live audio play (Vn)	This application will be running in two different cases, one is as a stand alone application representing a live stream video. The other case is when this. Application is synchronized with MPlayer live audio stream to represent audio/video movie play.
Wget tool (Wg)	This application is used to simultaneously download from one to x number of files from an online server in a fixed amount of time, where x is the maximum number of files that the system can successfully download at the same time.
Ice-weasel web-browser (Iw)	This application is used to web-browse from one to x on-line web sites sequentially separated by a fixed time interval, where x is the maximum number of websites that the system can successfully open at the same time.
Dhrystone's benchmark (Dh)	This application focuses on string handling, as there are no floating point operations. It is heavily influenced by hardware and software. design, compiler and linker options, code optimization, cache memory, wait states, and integer data types.
WhestStone benchmark (Wh)	This test contains several modules that are meant to represent a mix of operations typically performed in scientific applications. A wide variety of C functions including sin, cos, sqrt, exp, and log are used as well as floating-point math operations, array accesses, conditional as an integer and branches, and procedure calls. This test measures both integer and floating-point arithmetic.

– Hash miss and physical address hit.

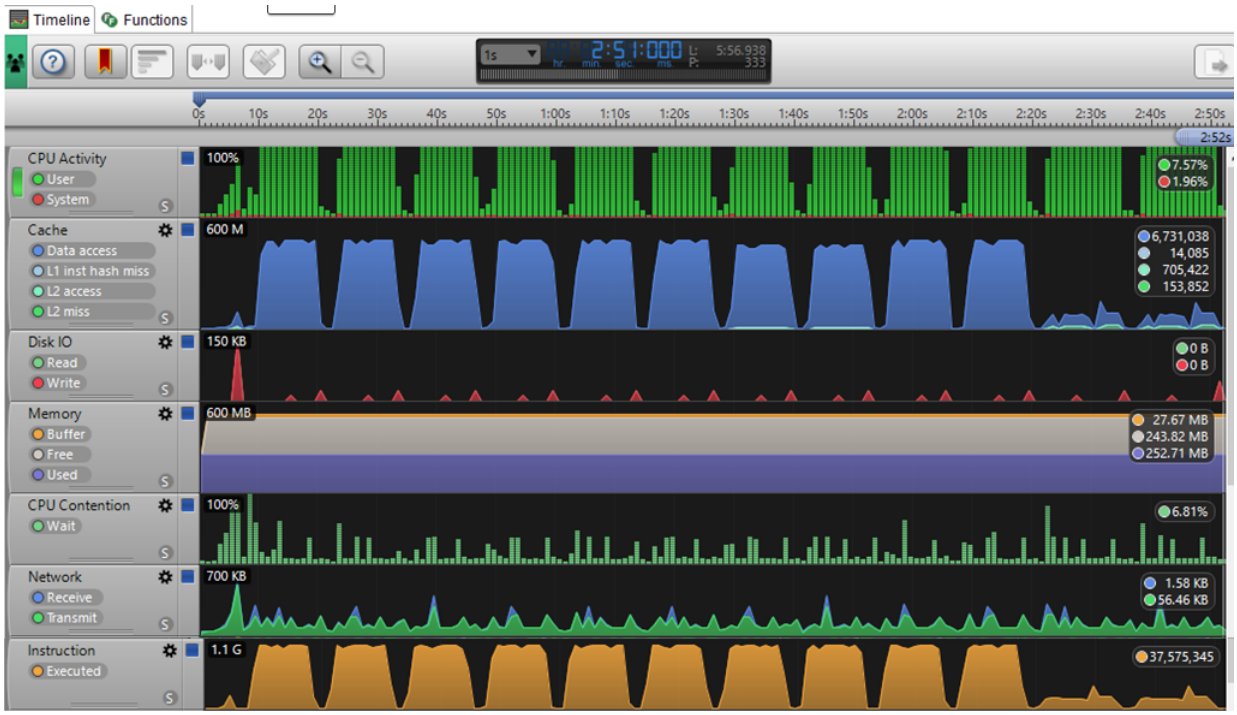

Figure 6.2 DS-5 streaming Counters used to measure system performance.

By using the previous 5 counters we are able to estimate the traffic generated by each application, calculated as follows:

– **Traffic on shared memory** = *L2 miss (0x44)*
– **Traffic on L2 Cache** = The number of L2 hits = *Data Access—L2 miss (0x44)*
– **Traffic on L1 Cache** = The number of L1 instruction hits + the number of L1 Cache hits = *(Instruction Executed [0x08]—L1 inst Hash miss [0x4a]) + (Data Access—L2 Access [0x43])*
– **Traffic on Processing unit** = It should be the same as the traffic on L1 Cache = the number of L1 instruction hits + the number of L1 Cache hits = *(Instruction Executed [0x08]—L1 inst Hash miss [0x4a]) + (Data Access—L2 Access [0x43])*

The estimate of traffic on the input and output component is calculated differently, based on the type of application (see Table 6.2 for more details). For this purpose, we use 4 more counters from the streamline DS-5 analyzer:

– **Disk IO read counter**: Disk IO Bytes Read.
– **Disk IO write counter**: Disk IO Bytes Written.
– **Network receive counter**: Receive network traffic, including the effect from streamline

– **Network transmit counter**: Transmit network traffic, including the effect from streamline

We try to minimize the cases where we use the previous counters to calculate the input and output devices' traffic. This is because these counters are for all reads and writes on the system and cannot be isolated for one application.

To use it, we try to take the counters reading when only the operating system running. And we take the reading again while the application running, Capture the difference and use it. Another way is to

Alternatively, we replace these reading by the reasing taken from application itself. For example, Mplayer have benchmark command line option provide such information.

6.1.2 Decision Variables

Each job (e.g. j_i^{al}) is further loosen up into a set of tasks (e.g. T_i^{al}), one task per possible resource in the resource chain. This number is the same as the number of jobs needed by each application have to run through the system in order to be complete.

Furthermore, each task on each component is decompressed into a set of subtasks. The number of these subtasks changes based on the maximum capacity of the resource and the minimum unit of time needed to handle this task.

Note that the number of tasks on each resource changes from one resource to the next. We denote this number by λ_i^x where i is the task location in the resource chain for application x.

Since the BBB has only one processing core, no alternate resource is considered in the application resource chains. Of course this can be changed easily by adding a variable to represent the number of available cores in the system and apply the alternative task option as we did in the model described in Chapter 5.

Our set of decision variables is then:

$$T = \bigcup_{1\le i\le\varphi_{al}} T_i^{al} \cup \bigcup_{1\le i\le\varphi_{vl}} T_i^{vl} \cup \bigcup_{1\le i\le\varphi_{an}} T_i^{an} \cup \bigcup_{1\le i\le\varphi_{vn}} T_i^{vn} \cup \bigcup_{1\le i\le\varphi_{wg}} T_i^{wg} \cup \bigcup_{1\le i\le\varphi_{iw}} T_i^{iw} \cup \bigcup_{1\le i\le\varphi_{dh}} T_i^{dh} \cup \bigcup_{1\le i\le\varphi_{wh}} T_i^{wh} \tag{6.1}$$

where

Table 6.2 Devices input and traffic calculation for the different applications running in the system

Application name	Input device	Input device Calculations	Output device	Output device calculations
(Al)	HD	Disk IO read counter	AuP	benchmark command-line option in MPlayer application
(Vl)	HD	Disk IO read counter	GCr	benchmark command-line option in MPlayer application
(An)	RJ	Network receive counter	AuP	benchmark command-line option in MPlayer application
(Vn)	RJ	Network receive counter	GCr	benchmark command-line option in MPlayer application
(Wg)	RJ	Network receive counter	HD	Disk IO write counter
(Iw)	RJ	Network receive counter	HD	Disk IO write counter
(Dh)	HD	Disk IO read counter	HD	Disk IO write counter
(Wh)	HD	Disk IO read counter	HD	Disk IO write counter

$$
\begin{aligned}
T_i^{al} &= \{t_{i,k}^{al} : 1 \leq k \leq \lambda_i^{al},\ 1 \leq i \leq \varphi_{al}\} \\
T_i^{vl} &= \{t_{i,k}^{vl} : 1 \leq k \leq \lambda_i^{vl},\ 1 \leq i \leq \varphi_{vl}\} \\
T_i^{an} &= \{t_{i,k}^{an} : 1 \leq k \leq \lambda_i^{an},\ 1 \leq i \leq \varphi_{an}\} \\
T_i^{vn} &= \{t_{i,k}^{vn} : 1 \leq k \leq \lambda_i^{vn},\ 1 \leq i \leq \varphi_{vn}\} \\
T_i^{wg} &= \{t_{i,k}^{wg} : 1 \leq k \leq \lambda_i^{wg} * noF,\ 1 \leq i \leq \varphi_{wg}\} \\
T_i^{iw} &= \{t_{i,k}^{iw} : 1 \leq k \leq \lambda_i^{iw} * noW,\ 1 \leq i \leq \varphi_{iw}\} \\
T_i^{dh} &= \{t_{i,k}^{dh} : 1 \leq k \leq \lambda_i^{dh},\ 1 \leq i \leq \varphi_{dh}\} \\
T_i^{wh} &= \{t_{i,k}^{wh} : 1 \leq k \leq \lambda_i^{wh},\ 1 \leq i \leq \varphi_{wh}\}
\end{aligned} \tag{6.2}
$$

Note that there are two more parameters noF and noW, considered with Wg and Iw, respectively.

This is because these two applications will have to run more than once through the system try to cause the system to fail.

As usual for each task, we associate the $start, end$ times when the task starts and ends

being processed on the corresponding resource respectively, *demand* (the space it occupies on the component while being processed), and *presence*, which indicates whether the task is currently scheduled.

6.1.3 Constraints

As with the previous models, we have 4 main sets of constraints controlling capacity, duration, scheduling start and end time, and dependency. Also, we added existence group, a group of constraints to separate the optional from the mandatory tasks. The constraints are chosen to ensure both system and application rules are respected. In here we did not list all constraints. We gave example of each one used.

Existence Constraints This constraint enforces the existence or absence of some of the tasks on the system. It is represented by the variable *presence*. If the task exists the variable is set to $True$ — otherwise, it will be $False$.

Constraint 1: Once we decide which application should be running into the system, all its tasks must be scheduled and exist on the model. We are using the parameter ε to represent if the application is or is not running on the system. We will set all tasks *presence* value to True if its corresponding ε parameter is true.

$$t^{al}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{al}, \quad \varepsilon_{al} = True, \quad 1 \leq k \leq \lambda^{al}_i \tag{6.3}$$

$$t^{vl}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{vl}, \quad \varepsilon_{vl} = True, \quad 1 \leq k \leq \lambda^{vl}_i \tag{6.4}$$

$$t^{an}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{an}, \quad \varepsilon_{an} = True, \quad 1 \leq k \leq \lambda^{an}_i \tag{6.5}$$

$$t^{vn}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{vn}, \quad \varepsilon_{vn} = True, \quad 1 \leq k \leq \lambda^{vn}_i \tag{6.6}$$

$$t^{wg}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{wg}, \quad \varepsilon_{wg} = True, \quad 1 \leq k \leq \lambda^{wg}_i * noF \tag{6.7}$$

$$t^{iw}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{iw}, \quad \varepsilon_{iw} = True, \quad 1 \leq k \leq \lambda^{iw}_i * noW \tag{6.8}$$

$$t^{dh}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{dh}, \quad \varepsilon_{dh} = True, \quad 1 \leq k \leq \lambda^{dh}_i \tag{6.9}$$

$$t^{wh}_{i,k}.presence = True, \quad Where \quad 1 \leq i \leq \varphi_{wh}, \quad \varepsilon_{wh} = True, \quad 1 \leq k \leq \lambda^{wh}_i \tag{6.10}$$

Capacity Constraints Given D the simulation deadline, these constraints specify the different resources needed by different tasks and the allowed maximum capacity for the constraints.

Constraint 2: Resource capacity must be respected at all times. Note that ILOG solver use specialized, optimized function named “cumulFunction” and “pulse” to handle such

resource usage constraints globally. This was explained more in Chapter 2.

Each of the 8 applications follows a various ordered chain of resources depending on the application input and output. This ordered chain is denoted by ω_i^x where $1 \leq i \leq \varphi_x$.

$$\begin{aligned}
\omega^{al} &= \{HD, LC_2, LC_1, PE, AuP\}, \quad \omega^{vl} = \{HD, LC_2, LC_1, PE, GCr\},\\
\omega^{an} &= \{RJ, LC_2, LC_1, PE, AuP\}, \quad \omega^{vn} = \{RJ, LC_2, LC_1, PE, GCr\},\\
\omega^{wg} &= \{RJ, LC_2, LC_1, PE, HD\}, \quad \omega^{iw} = \{RJ, LC_2, LC_1, PE, HD\},\\
\omega^{dh} &= \{HD, LC_2, LC_1, PE, HD\}, \quad \omega^{wh} = \{HD, LC_2, LC_1, PE, HD\}
\end{aligned} \tag{6.11}$$

Depending on the resource chain flow of each application the following constraints apply:

$$\begin{aligned}
RJ.Capacity \geq\ & \sum_{1\leq k\leq \lambda_1^{al}} (t_{1,k}^{al}.demand * t_{1,k}^{al}.presence) + \sum_{1\leq k\leq \lambda_{\varphi al}^{al}} (t_{\varphi_{al},k}^{al}.demand * t_{\varphi_{al},k}^{al}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{vl}} (t_{1,k}^{vl}.demand * t_{1,k}^{vl}.presence) + \sum_{1\leq k\leq \lambda_{\varphi vl}^{vl}} (t_{\varphi_{vl},k}^{vl}.demand * t_{\varphi_{vl},k}^{vl}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{an}} (t_{1,k}^{an}.demand * t_{1,k}^{an}.presence) + \sum_{1\leq k\leq \lambda_{\varphi an}^{an}} (t_{\varphi_{an},k}^{an}.demand * t_{\varphi_{an},k}^{an}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{vn}} (t_{1,k}^{vn}.demand * t_{1,k}^{vn}.presence) + \sum_{1\leq k\leq \lambda_{\varphi vn}^{vn}} (t_{\varphi_{vn},k}^{vn}.demand * t_{\varphi_{vn},k}^{vn}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{wg} * noF} (t_{1,k}^{wg}.demand * t_{1,k}^{wg}.presence) + \sum_{1\leq k\leq \lambda_{\varphi wg}^{wg} * noF} (t_{\varphi_{wg},k}^{wg}.demand * t_{\varphi_{wg},k}^{wg}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{iw} * noW} (t_{1,k}^{iw}.demand * t_{1,k}^{iw}.presence) + \sum_{1\leq k\leq \lambda_{\varphi iw}^{iw} * noW} (t_{\varphi_{iw},k}^{iw}.demand * t_{\varphi_{iw},k}^{iw}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{dh}} (t_{1,k}^{dh}.demand * t_{1,k}^{dh}.presence) + \sum_{1\leq k\leq \lambda_{\varphi dh}^{dh}} (t_{\varphi_{dh},k}^{dh}.demand * t_{\varphi_{dh},k}^{dh}.presence) +\\
& \sum_{1\leq k\leq \lambda_1^{wh}} (t_{1,k}^{wh}.demand * t_{1,k}^{wh}.presence) + \sum_{1\leq k\leq \lambda_{\varphi wh}^{wh}} (t_{\varphi_{wh},k}^{wh}.demand * t_{\varphi_{wh},k}^{wh}.presence) +
\end{aligned}$$

Where $0 \leq d \leq D$ **such that** :

$$\begin{aligned}
&t_{1,k}^{al}.start \leq d \leq t_{1,k}^{al}.end \ \textbf{And}\ t_{\varphi_{al},k}^{al}.start \leq d \leq t_{\varphi_{al},k}^{al}.end \ \textbf{Given}\ \omega_1^{al} = RJ \ \textbf{And}\ \omega_{\varphi_{al}}^{al} = RJ,\\
&t_{1,k}^{vl}.start \leq d \leq t_{1,k}^{vl}.end \ \textbf{And}\ t_{\varphi_{vl},k}^{vl}.start \leq d \leq t_{\varphi_{vl},k}^{vl}.end \ \textbf{Given}\ \omega_1^{vl} = RJ \ \textbf{And}\ \omega_{\varphi_{vl}}^{vl} = RJ,\\
&t_{1,k}^{an}.start \leq d \leq t_{1,k}^{an}.end \ \textbf{And}\ t_{\varphi_{an},k}^{an}.start \leq d \leq t_{\varphi_{an},k}^{an}.end \ \textbf{Given}\ \omega_1^{an} = RJ \ \textbf{And}\ \omega_{\varphi_{an}}^{an} = RJ,\\
&t_{1,k}^{vn}.start \leq d \leq t_{1,k}^{vn}.end \ \textbf{And}\ t_{\varphi_{vn},k}^{vn}.start \leq d \leq t_{\varphi_{vn},k}^{vn}.end \ \textbf{Given}\ \omega_1^{vn} = RJ \ \textbf{And}\ \omega_{\varphi_{vn}}^{vn} = RJ,\\
&t_{1,k}^{wg}.start \leq d \leq t_{1,k}^{wg}.end \ \textbf{And}\ t_{\varphi_{wg},k}^{wg}.start \leq d \leq t_{\varphi_{wg},k}^{wg}.end \ \textbf{Given}\ \omega_1^{wg} = RJ \ \textbf{And}\ \omega_{\varphi_{wg}}^{wg} = RJ,\\
&t_{1,k}^{iw}.start \leq d \leq t_{1,k}^{iw}.end \ \textbf{And}\ t_{\varphi_{iw},k}^{iw}.start \leq d \leq t_{\varphi_{iw},k}^{iw}.end \ \textbf{Given}\ \omega_1^{iw} = RJ \ \textbf{And}\ \omega_{\varphi_{iw}}^{iw} = RJ,\\
&t_{1,k}^{dh}.start \leq d \leq t_{1,k}^{dh}.end \ \textbf{And}\ t_{\varphi_{dh},k}^{dh}.start \leq d \leq t_{\varphi_{dh},k}^{dh}.end \ \textbf{Given}\ \omega_1^{dh} = RJ \ \textbf{And}\ \omega_{\varphi_{dh}}^{dh} = RJ,\\
&t_{1,k}^{wh}.start \leq d \leq t_{1,k}^{wh}.end \ \textbf{And}\ t_{\varphi_{wh},k}^{wh}.start \leq d \leq t_{\varphi_{wh},k}^{wh}.end \ \textbf{Given}\ \omega_1^{wh} = RJ \ \textbf{And}\ \omega_{\varphi_{wh}}^{wh} = RJ,
\end{aligned} \tag{6.12}$$

In the previous equation we consider only the first and last jobs in the resource chain. This is because the RJ component is only used as an input or output device (see Table 6.2). The same equation can be applied to the HD resource. Note that d here represent time in the equation.

$$
\begin{aligned}
HD.Capacity \geq \quad & \sum_{1\leq k\leq \lambda_1^{al}} (t_{1,k}^{al}.demand * t_{1,k}^{al}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{al}}^{al}} (t_{\varphi_{al},k}^{al}.demand * t_{\varphi_{al},k}^{al}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{vl}} (t_{1,k}^{vl}.demand * t_{1,k}^{vl}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{vl}}^{vl}} (t_{\varphi_{vl},k}^{vl}.demand * t_{\varphi_{vl},k}^{vl}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{an}} (t_{1,k}^{an}.demand * t_{1,k}^{an}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{an}}^{an}} (t_{\varphi_{an},k}^{an}.demand * t_{\varphi_{an},k}^{an}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{vn}} (t_{1,k}^{vn}.demand * t_{1,k}^{vn}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{vn}}^{vn}} (t_{\varphi_{vn},k}^{vn}.demand * t_{\varphi_{vn},k}^{vn}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{wg} * noF} (t_{1,k}^{wg}.demand * t_{1,k}^{wg}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{wg}}^{wg} * noF} (t_{\varphi_{wg},k}^{wg}.demand * t_{\varphi_{wg},k}^{wg}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{iw} * noW} (t_{1,k}^{iw}.demand * t_{1,k}^{iw}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{iw}}^{iw} * noW} (t_{\varphi_{iw},k}^{iw}.demand * t_{\varphi_{iw},k}^{iw}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{dh}} (t_{1,k}^{dh}.demand * t_{1,k}^{dh}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{dh}}^{dh}} (t_{\varphi_{dh},k}^{dh}.demand * t_{\varphi_{dh},k}^{dh}.presence) + \\
& \sum_{1\leq k\leq \lambda_1^{wh}} (t_{1,k}^{wh}.demand * t_{1,k}^{wh}.presence) + \sum_{1\leq k\leq \lambda_{\varphi_{wh}}^{wh}} (t_{\varphi_{wh},k}^{wh}.demand * t_{\varphi_{wh},k}^{wh}.presence) +
\end{aligned}
$$

Where $0 \leq d \leq D$ **such that** :

$$
\begin{aligned}
& t_{1,k}^{al}.start \leq \quad d \leq t_{1,k}^{al}.end \quad \textbf{And} \quad t_{\varphi_{al},k}^{al}.start \leq t \leq t_{\varphi_{al},k}^{al}.end \quad \textbf{Given} \quad \omega_1^{al} = HD \quad \textbf{And} \quad \omega_{\varphi_{al}}^{al} = HD, \\
& t_{1,k}^{vl}.start \leq \quad d \leq t_{1,k}^{vl}.end \quad \textbf{And} \quad t_{\varphi_{vl},k}^{vl}.start \leq d \leq t_{\varphi_{vl},k}^{vl}.end \quad \textbf{Given} \quad \omega_1^{vl} = HD \quad \textbf{And} \quad \omega_{\varphi_{vl}}^{vl} = HD, \\
& t_{1,k}^{an}.start \leq \quad d \leq t_{1,k}^{an}.end \quad \textbf{And} \quad t_{\varphi_{an},k}^{an}.start \leq d \leq t_{\varphi_{an},k}^{an}.end \quad \textbf{Given} \quad \omega_1^{an} = HD \quad \textbf{And} \quad \omega_{\varphi_{an}}^{an} = HD, \\
& t_{1,k}^{vn}.start \leq \quad d \leq t_{1,k}^{vn}.end \quad \textbf{And} \quad t_{\varphi_{vn},k}^{vn}.start \leq d \leq t_{\varphi_{vn},k}^{vn}.end \quad \textbf{Given} \quad \omega_1^{vn} = HD \quad \textbf{And} \quad \omega_{\varphi_{vn}}^{vn} = HD, \\
& t_{1,k}^{wg}.start \leq \quad d \leq t_{1,k}^{wg}.end \quad \textbf{And} \quad t_{\varphi_{wg},k}^{wg}.start \leq d \leq t_{\varphi_{wg},k}^{wg}.end \quad \textbf{Given} \quad \omega_1^{wg} = HD \quad \textbf{And} \quad \omega_{\varphi_{wg}}^{wg} = HD, \\
& t_{1,k}^{iw}.start \leq \quad d \leq t_{1,k}^{iw}.end \quad \textbf{And} \quad t_{\varphi_{iw},k}^{iw}.start \leq d \leq t_{\varphi_{iw},k}^{iw}.end \quad \textbf{Given} \quad \omega_1^{iw} = HD \quad \textbf{And} \quad \omega_{\varphi_{iw}}^{iw} = HD, \\
& t_{1,k}^{dh}.start \leq \quad d \leq t_{1,k}^{dh}.end \quad \textbf{And} \quad t_{\varphi_{dh},k}^{dh}.start \leq d \leq t_{\varphi_{dh},k}^{dh}.end \quad \textbf{Given} \quad \omega_1^{dh} = HD \quad \textbf{And} \quad \omega_{\varphi_{dh}}^{dh} = HD, \\
& t_{1,k}^{wh}.start \leq \quad d \leq t_{1,k}^{wh}.end \quad \textbf{And} \quad t_{\varphi_{wh},k}^{wh}.start \leq d \leq t_{\varphi_{wh},k}^{wh}.end \quad \textbf{Given} \quad \omega_1^{wh} = HD \quad \textbf{And} \quad \omega_{\varphi_{wh}}^{wh} = HD,
\end{aligned}
\tag{6.13}
$$

For the next four resources (SM, LC_2, LC_1, PE) the application runs through them only once in a fixed order: $SM- > LC_2- > LC_1- > PE$, starting from the second to the fifth location. Getting back to Figure 6.1, we can see the traffics assumed to be uni-direction. Since we measure the traffics from the counters read fro DS-5, it is safe to assume the applications traffic is uni-direction.

$$
\begin{aligned}
SM.Capacity \geq \quad & \sum_{1\leq k\leq \lambda_2^{al}} (t_{2,k}^{al}.demand * t_{2,k}^{al}.presence) + \sum_{1\leq k\leq \lambda_2^{vl}} (t_{2,k}^{vl}.demand * t_{2,k}^{vl}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_2^{an}} (t_{2,k}^{an}.demand * t_{2,k}^{an}.presence) + \sum_{1\leq k\leq \lambda_2^{vn}} (t_{2,k}^{vn}.demand * t_{2,k}^{vn}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_2^{wg} * noF} (t_{2,k}^{wg}.demand * t_{2,k}^{wg}.presence) + \sum_{1\leq k\leq \lambda_2^{iw} * noW} (t_{2,k}^{iw}.demand * t_{2,k}^{iw}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_2^{dh}} (t_{2,k}^{dh}.demand * t_{2,k}^{dh}.presence) + \sum_{1\leq k\leq \lambda_2^{wh}} (t_{2,k}^{wh}.demand * t_{2,k}^{wh}.presence)
\end{aligned}
$$

$$
\begin{aligned}
Where \qquad & 0 \leq d \leq D \quad such \quad that: \\
& t_{2,k}^{al}.start \leq d \leq t_{2,k}^{al}.end \quad And \quad \omega_2^{al} = SM, \\
& t_{2,k}^{vl}.start \leq d \leq t_{2,k}^{vl}.end \quad And \quad \omega_2^{vl} = SM, \\
& t_{2,k}^{an}.start \leq d \leq t_{2,k}^{an}.end \quad And \quad \omega_2^{an} = SM, \\
& t_{2,k}^{vn}.start \leq d \leq t_{2,k}^{vn}.end \quad And \quad \omega_2^{vn} = SM, \\
& t_{2,k}^{wg}.start \leq d \leq t_{2,k}^{wg}.end \quad And \quad \omega_2^{wg} = SM, \\
& t_{2,k}^{iw}.start \leq d \leq t_{2,k}^{iw}.end \quad And \quad \omega_2^{iw} = SM, \\
& t_{2,k}^{dh}.start \leq d \leq t_{2,k}^{dh}.end \quad And \quad \omega_2^{dh} = SM, \\
& t_{2,k}^{wh}.start \leq d \leq t_{2,k}^{wh}.end \quad And \quad \omega_2^{wh} = SM,
\end{aligned}
\tag{6.14}
$$

$$
\begin{aligned}
LC_2.Capacity \geq \quad & \sum_{1\leq k\leq \lambda_3^{al}} (t_{3,k}^{al}.demand * t_{3,k}^{al}.presence) + \sum_{1\leq k\leq \lambda_3^{vl}} (t_{3,k}^{vl}.demand * t_{3,k}^{vl}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_3^{an}} (t_{3,k}^{an}.demand * t_{3,k}^{an}.presence) + \sum_{1\leq k\leq \lambda_3^{vn}} (t_{3,k}^{vn}.demand * t_{3,k}^{vn}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_3^{wg} * noF} (t_{3,k}^{wg}.demand * t_{3,k}^{wg}.presence) + \sum_{1\leq k\leq \lambda_3^{iw} * noW} (t_{3,k}^{iw}.demand * t_{3,k}^{iw}.presence) \quad + \\
& \sum_{1\leq k\leq \lambda_3^{dh}} (t_{3,k}^{dh}.demand * t_{3,k}^{dh}.presence) + \sum_{1\leq k\leq \lambda_3^{wh}} (t_{3,k}^{wh}.demand * t_{3,k}^{wh}.presence)
\end{aligned}
$$

$$
\begin{aligned}
Where \qquad & 0 \leq d \leq D \quad such \quad that: \\
& t_{3,k}^{al}.start \leq d \leq t_{3,k}^{al}.end \quad And \quad \omega_3^{al} = LC_2, \\
& t_{3,k}^{vl}.start \leq d \leq t_{3,k}^{vl}.end \quad And \quad \omega_3^{vl} = LC_2, \\
& t_{3,k}^{an}.start \leq d \leq t_{3,k}^{an}.end \quad And \quad \omega_3^{an} = LC_2, \\
& t_{3,k}^{vn}.start \leq d \leq t_{3,k}^{vn}.end \quad And \quad \omega_3^{vn} = LC_2, \\
& t_{3,k}^{wg}.start \leq d \leq t_{3,k}^{wg}.end \quad And \quad \omega_3^{wg} = LC_2, \\
& t_{3,k}^{iw}.start \leq d \leq t_{3,k}^{iw}.end \quad And \quad \omega_3^{iw} = LC_2, \\
& t_{3,k}^{dh}.start \leq d \leq t_{3,k}^{dh}.end \quad And \quad \omega_3^{dh} = LC_2, \\
& t_{3,k}^{wh}.start \leq d \leq t_{3,k}^{wh}.end \quad And \quad \omega_3^{wh} = LC_2,
\end{aligned}
\tag{6.15}
$$

The same equation applied to LC_1 and PE with there corresponding locations.

Duration Constraints Specify the processing duration for different tasks.

Constraint 3: The task duration should be bigger than or equal to the time a certain resource needs to process the application packet according to both resource speed and task size.

The task size or duration is changeable from one resource to the next. The task duration depending on the Max resource capacity and the minimum unit time can be used along with the application properties.

Some of the streaming applications constraints should be satisfied in one second (e.g. number of displayed frames per second), so we prefer to use the traffic generated per second by each of the applications and use it to calculate the task duration. The start of each of the sequential tasks is separated by one second.

It means that the number of tasks per application will be equal to the number of seconds the application spends in the system. For example, if we have an application runs-time equal 13 second, we will start with considering 13 subtasks needed in each resource in this application chain of resource to complete all tasks related to this application and consider it run through the system.

Then we calculate the subtask duration by dividing the traffic generated on the resource in one second by the resource speed. And we force separation between tasks with one second.

Unfortunately, this is not possible in all cases as the traffic generated in one second for some subtasks takes a tiny fraction of time can not be represented in a microsecond unit. This is why further calculations are needed to get the most suitable number of tasks based on task duration. We need to find the minimum subtask duration run on the resource. And at the same time this traffic is not exceeded the resource capacity.

Each of the applications has a total traffic generated per component, a total duration equal to the total application run-time in the system, the initial task time which is the theoretical proposed task time based on the nature of the application and, total traffic duration which is the actual time the application needed the resource to finish its tasks. This is denoted by $Ttraffic$, $Duration$, $Tduration$ and, $TTDuration$ respectively. For example, making application Al running on the PE resource:

$$Al.TTDuration_{PE} = \lceil \frac{Al.Ttraffic_{PE}}{PE.speed} \rceil$$

$$\lambda^{al}_{\omega^{al}_4 = PE} = \lceil \frac{Al.TTDuration_{PE}}{Al.Tduration_{PE}} \rceil$$

Note that we choose the task duration as the minimum possible time value representable in the system. And at the same time, it does not violate the total application traffic duration

and bigger than one time unit.

$$t^{al}_{\omega^{al}_4=PE,j}.presence = True \Rightarrow t^{al}_{\omega^{al}_4=PE,j}.duration \geq Al.Tduration_{PE},\ 1 \leq j \leq \lambda^{al}_{\omega^{al}_4=PE}$$

The same constraint is applied to every component in each of the applications chain of resources.

Scheduling start and end time Constraints Specify the processing start and end time for different tasks.

Constraint 4: tasks must end processing before the simulation deadline D. Here we need only to restrict the end of the last resource since packets flow in the system sequentially.

$$\begin{aligned}
t^{al}_{\varphi_{al},j}.end \leq D,\quad 1 \leq j \leq \lambda^{al}_{\varphi_{al}} \\
t^{vl}_{\varphi_{vl},j}.end \leq D,\quad 1 \leq j \leq \lambda^{vl}_{\varphi_{vl}} \\
t^{an}_{\varphi_{an},j}.end \leq D,\quad 1 \leq j \leq \lambda^{an}_{\varphi_{an}} \\
t^{vn}_{\varphi_{vn},j}.end \leq D,\quad 1 \leq j \leq \lambda^{vn}_{\varphi_{vn}} \\
t^{wg}_{\varphi_{wg},j}.end \leq D,\quad 1 \leq j \leq \lambda^{wg}_{\varphi_{wg}} * noF \\
t^{iw}_{\varphi_{iw},j}.end \leq D,\quad 1 \leq j \leq \lambda^{iw}_{\varphi_{iw}} * noW \\
t^{dh}_{\varphi_{dh},j}.end \leq D,\quad 1 \leq j \leq \lambda^{dh}_{\varphi_{dh}} \\
t^{wh}_{\varphi_{wh},j}.end \leq D,\quad 1 \leq j \leq \lambda^{wh}_{\varphi_{wh}}
\end{aligned} \tag{6.16}$$

Constraint 5: Some applications restrict the maximum delay allowed by the system buffer or any other reason before it actually starts running. Each of the applications has a parameter to define the maximum start delay allowed. This parameter denoted by *startDelay*. This is restricted only to the first task run on the processor core PE per application.

$$
\begin{aligned}
t^{al}_{\omega^{al}_4=PE,1}.start &\leq Al.startDelay \\
t^{vl}_{\omega^{vl}_4=PE,1}.start &\leq Vl.startDelay \\
t^{an}_{\omega^{an}_4=PE,1}.start &\leq An.startDelay \\
t^{vn}_{\omega^{vn}_4=PE,1}.start &\leq Vn.startDelay \\
t^{wg}_{\omega^{wg}_4=PE,1}.start &\leq Wg.startDelay \\
t^{iw}_{\omega^{iw}_4=PE,1}.start &\leq Iw.startDelay \\
t^{dh}_{\omega^{dh}_4=PE,1}.start &\leq Dh.startDelay \\
t^{wh}_{\omega^{wh}_4=PE,1}.start &\leq Wh.startDelay
\end{aligned}
\tag{6.17}
$$

Note that here the noF and noW did not affect this constraint as the running of all these applications is sequential. We need only to restrict the first task of the first website or file.

Constraint 6: All tasks are processed in sequential order on the same component.

$$
\begin{aligned}
t^{al}_{i,j}.start \leq t^{al}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{al}\ \ and\ \ 1 \leq j \leq \lambda^{al}_i - 1 \\
t^{av}_{i,j}.start \leq t^{av}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{av}\ \ and\ \ 1 \leq j \leq \lambda^{av}_i - 1 \\
t^{an}_{i,j}.start \leq t^{an}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{an}\ \ and\ \ 1 \leq j \leq \lambda^{an}_i - 1 \\
t^{vn}_{i,j}.start \leq t^{vn}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{vn}\ \ and\ \ 1 \leq j \leq \lambda^{vn}_i - 1 \\
t^{wg}_{i,j}.start \leq t^{wg}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{wg}\ \ and\ \ 1 \leq j \leq (\lambda^{wg}_i * noF) - 1 \\
t^{iw}_{i,j}.start \leq t^{iw}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{iw}\ \ and\ \ 1 \leq j \leq (\lambda^{iw}_i * noW) - 1 \\
t^{dh}_{i,j}.start \leq t^{dh}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{dh}\ \ and\ \ 1 \leq j \leq \lambda^{dh}_i - 1 \\
t^{wh}_{i,j}.start \leq t^{wh}_{i,j+1}.start,\ 1 \leq i \leq \varphi_{wh}\ \ and\ \ 1 \leq j \leq \lambda^{wh}_i - 1
\end{aligned}
\tag{6.18}
$$

Constraint 7: One task cannot start on the next resource on its application resource chain before it spends the time required processing at least one packet on its current resource. Also one task must start on its next resource in the application resource chain once the time required to process tasks on the current resource reaches the resource's maximum

capacity. To insure these two constraints each of the resources is assigned two parameters, $minProcessTime$ and $maxProcessTime$ to represent the time required to process traffic to fill one location into the resource and time require to completely fill this resource.

Because the number of tasks differs from one resource to the next into the application chain resource, there will be three different cases:

1. **Case one (the number of tasks in two sequence chains is the same):**
   For example: reviewing the model specifications you can notice that the PE unit and the L1-Cache will probably have the same number of tasks as they have almost the same speed and traffic load. The only difference is in the size of each of the resources.

$$
\begin{aligned}
& t^{al}_{LC_1,j}.start - t^{al}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{al}_{\omega^{al}_3=LC_1} = \lambda^{al}_{\omega^{al}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_3=LC_1} \\
& t^{vl}_{LC_1,j}.start - t^{vl}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{vl}_{\omega^{vl}_3=LC_1} = \lambda^{vl}_{\omega^{vl}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{vl}_{\omega^{vl}_3=LC_1} \\
& t^{an}_{LC_1,j}.start - t^{an}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{an}_{\omega^{an}_3=LC_1} = \lambda^{an}_{\omega^{an}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{an}_{\omega^{an}_3=LC_1} \\
& t^{vn}_{LC_1,j}.start - t^{vn}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{vn}_{\omega^{vn}_3=LC_1} = \lambda^{vn}_{\omega^{vn}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{vn}_{\omega^{vn}_3=LC_1} \\
& t^{wg}_{LC_1,j}.start - t^{wg}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{wg}_{\omega^{wg}_3=LC_1} = \lambda^{wg}_{\omega^{wg}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_3=LC_1} * noF \\
& t^{iw}_{LC_1,j}.start - t^{iw}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{iw}_{\omega^{iw}_3=LC_1} = \lambda^{iw}_{\omega^{iw}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{iw}_{\omega^{iw}_3=LC_1} * noW \\
& t^{dh}_{LC_1,j}.start - t^{dh}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{dh}_{\omega^{dh}_3=LC_1} = \lambda^{dh}_{\omega^{dh}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{dh}_{\omega^{dh}_3=LC_1} \\
& t^{wh}_{LC_1,j}.start - t^{wh}_{PE,j}.start \geq LC_1.minProcessTime, \\
& \quad Where\ \lambda^{wh}_{\omega^{wh}_3=LC_1} = \lambda^{wh}_{\omega^{wh}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{wh}_{\omega^{wh}_3=LC_1}
\end{aligned}
\tag{6.19}
$$

$$
\begin{aligned}
&t^{al}_{LC_1,j}.start - t^{al}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{al}_{\omega^{al}_3=LC_1} = \lambda^{al}_{\omega^{al}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_3=LC_1}\\
&t^{vl}_{LC_1,j}.start - t^{vl}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{vl}_{\omega^{vl}_3=LC_1} = \lambda^{vl}_{\omega^{vl}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{vl}_{\omega^{vl}_3=LC_1}\\
&t^{an}_{LC_1,j}.start - t^{an}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{an}_{\omega^{an}_3=LC_1} = \lambda^{an}_{\omega^{an}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{an}_{\omega^{an}_3=LC_1}\\
&t^{vn}_{LC_1,j}.start - t^{vn}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{vn}_{\omega^{vn}_3=LC_1} = \lambda^{vn}_{\omega^{vn}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{vn}_{\omega^{vn}_3=LC_1}\\
&t^{wg}_{LC_1,j}.start - t^{wg}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{wg}_{\omega^{wg}_3=LC_1} = \lambda^{wg}_{\omega^{wg}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_3=LC_1} * noF\\
&t^{iw}_{LC_1,j}.start - t^{iw}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{iw}_{\omega^{iw}_3=LC_1} = \lambda^{iw}_{\omega^{iw}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{iw}_{\omega^{iw}_3=LC_1} * noW\\
&t^{dh}_{LC_1,j}.start - t^{dh}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{dh}_{\omega^{dh}_3=LC_1} = \lambda^{dh}_{\omega^{dh}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{dh}_{\omega^{dh}_3=LC_1}\\
&t^{wh}_{LC_1,j}.start - t^{wh}_{PE,j}.start \leq LC_1.maxProcessTime,\\
&\quad Where\ \lambda^{wh}_{\omega^{wh}_3=LC_1} = \lambda^{wh}_{\omega^{wh}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{wh}_{\omega^{wh}_3=LC_1}
\end{aligned}
\tag{6.20}
$$

2. **Case two (the number of tasks in one resource is bigger than the next in two sequence chains):** For example: Assume the number of tasks needs to run on HD is bigger than the tasks run on SM then we need to ensure that the first task of the sequence running on the HD and the corresponding task run on SM respects the same minimum and maximum process time gap role.

$$
\begin{aligned}
&t^{al}_{SM,j}.start - t^{al}_{HD,1+(\frac{\lambda^{al}_{\omega^{al}_1=HD}}{\lambda^{al}_{\omega^{al}_2=SM}}*(j-1))}.start \geq HD.minProcessTime,\\
&\quad Where\ \lambda^{al}_{\omega^{al}_1=HD}\lambda^{al}_{\omega^{al}_2=SM}\ \ and\ \ \lambda^{al}_{\omega^{al}_1=HD}\mod \lambda^{al}_{\omega^{al}_2=SM} = 0\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_2=SM}
\end{aligned}
\tag{6.21}
$$

$$t^{al}_{SM,j}.start - t^{al}_{HD,1+(\frac{\lambda^{al}_{\omega^{al}_1=HD}}{\lambda^{al}_{\omega^{al}_2=SM}}*(j-1))}.start \leq HD.maxProcessTime,$$
$$Where\ \lambda^{al}_{\omega^{al}_1=HD}\lambda^{al}_{\omega^{al}_2=SM}\ \ and\ \ \lambda^{al}_{\omega^{al}_1=HD} \mod \lambda^{al}_{\omega^{al}_2=SM} = 0\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_2=SM} \tag{6.22}$$

The previous constraints applied to all applications except Wg and Iw as we should consider the amount of time the applications rerun into the system according to the number of files noF downloaded and the number of webs noW browsed.

$$t^{wg}_{SM,j+((i-1)*\lambda^{wg}_{\omega^{wg}_2=SM})}.start - t^{al}_{HD,1+(\frac{\lambda^{wg}_{\omega^{wg}_1=HD}}{\lambda^{wg}_{\omega^{al}_2=SM}}*(j-1)+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD}))}.start \geq HD.minProcessTime,$$
$$Where\ \lambda^{wg}_{\omega^{wg}_1=HD} > \lambda^{wg}_{\omega^{wg}_2=SM}\ \ and\ \ \lambda^{wg}_{\omega^{wg}_1=HD} \mod \lambda^{wg}_{\omega^{wg}_2=SM} = 0\ \ and$$
$$1 \leq i \leq noF\ \ and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_2=SM} \tag{6.23}$$

$$t^{wg}_{SM,j+((i-1)*\lambda^{wg}_{\omega^{wg}_2=SM})}.start - t^{al}_{HD,1+(\frac{\lambda^{wg}_{\omega^{wg}_1=HD}}{\lambda^{wg}_{\omega^{al}_2=SM}}*(j-1)+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD}))}.start \leq HD.maxProcessTime,$$
$$Where\ \lambda^{wg}_{\omega^{wg}_1=HD} > \lambda^{wg}_{\omega^{wg}_2=SM}\ \ and\ \ \lambda^{wg}_{\omega^{wg}_1=HD} \mod \lambda^{wg}_{\omega^{wg}_2=SM} = 0\ \ and$$
$$1 \leq i \leq noF\ \ and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_2=SM} \tag{6.24}$$

3. **Case three (the number of tasks in one resource is smaller than the next in two sequence chains):**
   For example: Assume the number of tasks needing to run on LC_2 is smaller than the tasks running on LC_1 then we need to ensure that the first task on the sequence running on the LC_1 and the corresponding task running on LC_2 respects the same minimum and maximum process time gap role.

$$t^{al}_{LC_1,(j-1)*\frac{\lambda^{al}_{\omega^{al}_4=LC_1}}{\lambda^{al}_{\omega^{al}_3=LC_2}}}.start - t^{al}_{LC_2,j}.start \geq LC_2.minProcessTime,$$
$$Where\ \lambda^{al}_{\omega^{al}_4=LC_1} > \lambda^{al}_{\omega^{al}_3=LC_2}\ \ and\ \ \lambda^{al}_{\omega^{al}_4=LC_1} \mod \lambda^{al}_{\omega^{al}_3=LC_2} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_3=LC_2} \tag{6.25}$$

$$t^{al}_{LC_1,(j-1)*\frac{\lambda^{al}_{\omega^{al}_4=LC_1}}{\lambda^{al}_{\omega^{al}_3=LC_2}}}.start - t^{al}_{LC_2,j}.start \leq LC_2.maxProcessTime,$$
$$Where\ \lambda^{al}_{\omega^{al}_4=LC_1} > \lambda^{al}_{\omega^{al}_3=LC_2}\ \ and\ \ \lambda^{al}_{\omega^{al}_4=LC_1} \mod \lambda^{al}_{\omega^{al}_3=LC_2} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_3=LC_2} \tag{6.26}$$

The previous constraints applied to all applications except Wg and Iw as we should consider the amount of time the applications rerun into the system according to the number of files noF downloaded and the number of webs noW browsed.

$$t^{wg}_{LC_1,(j-1)*\frac{\lambda^{wg}_{\omega^{wg}_4=LC_1}}{\lambda^{wg}_{\omega^{wg}_3=LC_2}}+((i-1)*\lambda^{wg}_{\omega^{wg}_4=LC_1})}.start - t^{wg}_{LC_2,j+(i-1)*\lambda^{wg}_{\omega^{wg}_3=LC_2}}.start \geq LC_2.minProcessTime,$$
$$Where\ \lambda^{wg}_{\omega^{wg}_4=LC_1} > \lambda^{wg}_{\omega^{wg}_3=LC_2}\ \ and\ \ \lambda^{wg}_{\omega^{wg}_4=LC_1} \mod \lambda^{wg}_{\omega^{wg}_3=LC_2} = 0\ \ and\ \ 1 \leq i \leq noF$$
$$and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_3=LC_2} \tag{6.27}$$

$$t^{wg}_{LC_1,(j-1)*\frac{\lambda^{wg}_{\omega^{wg}_4=LC_1}}{\lambda^{wg}_{\omega^{wg}_3=LC_2}}+((i-1)*\lambda^{wg}_{\omega^{wg}_4=LC_1})}.start - t^{wg}_{LC_2,j+(i-1)*\lambda^{wg}_{\omega^{wg}_3=LC_2}}.start \leq LC_2.maxProcessTime,$$
$$Where\ \lambda^{wg}_{\omega^{wg}_4=LC_1} > \lambda^{wg}_{\omega^{wg}_3=LC_2}\ \ and\ \ \lambda^{wg}_{\omega^{wg}_4=LC_1} \mod \lambda^{wg}_{\omega^{wg}_3=LC_2} = 0\ \ and\ \ 1 \leq i \leq noF$$
$$and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_3=LC_2} \tag{6.28}$$

Dependencies Constraints : Some of the tasks on one resource depend on the existence of other tasks on other resources on the application resource chain.

Constraint 8: Make sure that the task remains enough time in one resource until it is processed on the next one.

Because the number of tasks differs from one resource to the next in The application chain resource, this constraint will have three different cases:

1. **Case one (the number of tasks in two sequence chains is the same):**

$$t^{al}_{LC_1,j}.size \geq Al.Tduration_{PE},$$
$$Where\ \lambda^{al}_{\omega^{al}_3=LC_1} = \lambda^{al}_{\omega^{al}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_3=LC_1} \tag{6.29}$$

$$t^{wg}_{LC_1,j+((i-1)*\lambda^{wg}_{\omega^{wg}_4=LC_1})}.size \geq Wg.Tduration_{PE},$$
$$Where\ \lambda^{wg}_{\omega^{wg}_3=LC_1} = \lambda^{wg}_{\omega^{wg}_4=PE}\ \ and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_3=LC_1}\ \ and\ \ 1 \leq i \leq noF \tag{6.30}$$

2. **Case two (the number of tasks in one resource is bigger than the next in two sequence chains):**

$$t^{al}_{HD,\frac{\lambda^{al}_{\omega^{al}_1=HD}}{\lambda^{al}_{\omega^{al}_2=SM}}*j}.end - t^{al}_{HD,1+(\frac{\lambda^{al}_{\omega^{al}_1=HD}}{\lambda^{al}_{\omega^{al}_2=SM}}*(j-1))}.start \geq Al.Tduration_{SM},$$
$$Where\ \lambda^{al}_{\omega^{al}_1=HD} > \lambda^{al}_{\omega^{al}_2=SM}\ \ and\ \ \lambda^{al}_{\omega^{al}_1=HD} \mod \lambda^{al}_{\omega^{al}_2=SM} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_1=HD} \tag{6.31}$$

$$t^{wg}_{HD,(\frac{\lambda^{wg}_{\omega^{wg}_1=HD}}{\lambda^{wg}_{\omega^{wg}_2=SM}}*j)+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD})}.end - t^{wg}_{HD,(1+(\frac{\lambda^{wg}_{\omega^{wg}_1=HD}}{\lambda^{wg}_{\omega^{wg}_2=SM}}*(j-1)))+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD}))}.start \geq Al.Tduration_{SM},$$
$$Where\ \lambda^{wg}_{\omega^{wg}_1=HD} > \lambda^{wg}_{\omega^{wg}_2=SM}\ \ and\ \ \lambda^{al}_{\omega^{wg}_1=HD} \mod \lambda^{wg}_{\omega^{wg}_2=SM} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_1=HD} \tag{6.32}$$

3. **Case three (the number of tasks in one resource is smaller than the next in two sequence chains):**

$$t^{al}_{HD,\frac{\lambda^{al}_{\omega^{al}_2=SM}}{\lambda^{al}_{\omega^{al}_1=HD}}*j}.end - t^{al}_{HD,1+(\frac{\lambda^{al}_{\omega^{al}_1=HD}}{\lambda^{al}_{\omega^{al}_2=SM}}*(j-1))}.start \geq Al.Tduration_{SM},$$
$$Where\ \lambda^{al}_{\omega^{al}_1=HD} < \lambda^{al}_{\omega^{al}_2=SM}\ \ and\ \ \lambda^{al}_{\omega^{al}_2=SM} \mod \lambda^{al}_{\omega^{al}_1=HD} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_1=HD} \tag{6.33}$$

$$t^{wg}_{HD,(\frac{\lambda^{wg}_{\omega^{wg}_2=SM}}{\lambda^{wg}_{\omega^{wg}_1=HD}}*j)+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD})}.end - t^{wg}_{HD,(1+(\frac{\lambda^{wg}_{\omega^{wg}_1=HD}}{\lambda^{wg}_{\omega^{wg}_2=SM}}*(j-1)))+((i-1)*\lambda^{wg}_{\omega^{wg}_1=HD})}.start \geq Al.Tduration_{SM},$$
$$Where\ \lambda^{wg}_{\omega^{wg}_1=HD} < \lambda^{wg}_{\omega^{wg}_2=SM}\ \ and\ \ \lambda^{wg}_{\omega^{wg}_2=SM} \mod \lambda^{wg}_{\omega^{wg}_1=HD} = 0$$
$$and\ \ 1 \leq j \leq \lambda^{wg}_{\omega^{wg}_1=HD} \tag{6.34}$$

Per Application Constraints : Some constraints are application-specific.

Constraint 9: In case we run the audio/video stream we must make sure Audio/video tasks synchronize on the output resource.

$$t^{al}_{AuP,j}.start = t^{vl}_{GCr,j}.start$$
$$Where\ \lambda^{al}_{\omega^{al}_6=AuP} = \lambda^{vl}_{\omega^{al}_6=GCr}\ \ and\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_6=AuP} \tag{6.35}$$

Constraint 10: The number of audio and video output tasks is the same as the number of seconds the application actually runs. And their starts are separated by exactly one second.

$$t^{al}_{AuP,j+1}.start - t^{al}_{AuP,j}.start = 1Second$$
$$Where\ \ 1 \leq j \leq \lambda^{al}_{\omega^{al}_6=AuP} - 1 \tag{6.36}$$

Constraint 11: Statistically, users lose interest in the website they are browsing after the first five seconds if it does not have the information they are looking for. As a constraint we restricted the start of the first task in the processor core for each of the websites running by exactly five seconds.

$$t^{iw}_{PE,j+1}.start - t^{iw}_{PE,j}.start = 5Seconds$$
$$Where\ \ 1 \leq j \leq (\lambda^{al}_{\omega^{al}_5=PE} * noW) - 1\ \ and\ \ j \mod \lambda^{al}_{\omega^{al}_5=PE} = 1\ \ and\ \ \lambda^{al}_{\omega^{al}_5=PE} \neq 1 \tag{6.37}$$

$$t^{iw}_{PE,j+1}.start - t^{iw}_{PE,j}.start = 5Seconds$$
$$Where\ \ 1 \leq j \leq noW - 1\ \ and\ \ \lambda^{wg}_{\omega^{wg}_5=PE} = 1 \tag{6.38}$$

Constraint 12: The time between downloading different files through the Wg application is fixed and denoted by Γ_{wg}.

$$t^{wg}_{PE,j+1}.start - t^{wg}_{PE,j}.start = \Gamma_{wg}$$
$$Where \;\; 1 \leq j \leq (\lambda^{wg}_{\omega^{wg}_5=PE} * nof) - 1 \;\; and \;\; j \mod \lambda^{wg}_{\omega^{wg}_5=PE} = 1 \;\; and \;\; \lambda^{wg}_{\omega^{wg}_5=PE} \neq 1 \tag{6.39}$$

$$t^{wg}_{PE,j+1}.start - t^{iw}_{wg,j}.start = \Gamma_{wg}$$
$$Where \;\; 1 \leq j \leq noF - 1 \;\; and \;\; \lambda^{wg}_{\omega^{wg}_5=PE} = 1 \tag{6.40}$$

Constraint 13: As we can see in Figure 6.3 the *Dh* application is divided into 10 separate tasks. Each of the tasks runs in the processor for 11 seconds then writes to the disk for two seconds.

$$t^{dh}_{\omega^{dh}_6=HD,j+1}.start - t^{dh}_{\omega^{dh}_6=HD,j}.start = 13Second$$
$$Where \;\; 1 \leq j \leq \lambda^{dh}_{\omega^{dh}_6=HD} \tag{6.41}$$

$$t^{dh}_{\omega^{dh}_6=HD,j}.start - t^{dh}_{PE,(j-1)*(\frac{\lambda^{dh}_{\omega^{dh}_5=PE}}{\lambda^{dh}_{\omega^{dh}_6=HD}}+1)}.start = \;\; 11Seconds$$
$$Where \;\; 1 \leq j \leq \lambda^{dh}_{\omega^{dh}_6=HD} \;\; and$$
$$(\frac{\lambda^{dh}_{\omega^{dh}_5=PE}}{\lambda^{dh}_{\omega^{dh}_6=HD}}) * j \leq \lambda^{dh}_{\omega^{dh}_5=PE} \;\; and \;\; \lambda^{dh}_{\omega^{dh}_5=PE} \mod \lambda^{dh}_{\omega^{dh}_6=HD} = 0 \tag{6.42}$$

Constraint 14: As we can see in Figure 6.3 the *Dh* application first loads the whole code to memory then starts running. The same constraints apply to the *Wh* application (Figure 6.4).

$$t^{dh}_{HD,\lambda^{dh}_{\omega^{dh}_1=HD}}.start + Dh.Tduration_{HD} \geq t^{dh}_{SM,1}.start \tag{6.43}$$

$$t^{wh}_{HD,\lambda^{wh}_{\omega^{wh}_1=HD}}.start + Wh.Tduration_{HD} \geq t^{wh}_{SM,1}.start \tag{6.44}$$

Constraint 15: As we can see in Figure 6.3 the Wh application is divided into 10 separate tasks. Each of the tasks runs in the processor for 14 seconds then writes to the disk for two seconds.

$$t^{wh}_{\omega^{wh}_6=HD,j+1}.start - t^{wh}_{\omega^{wh}_6=HD,j}.start = 16Second$$
$$Where \;\; 1 \leq j \leq \lambda^{wh}_{\omega^{wh}_6=HD} \tag{6.45}$$

$$t^{wh}_{\omega^{wh}_6=HD,j}.start - t^{wh}_{PE,(j-1)*(\frac{\lambda^{wh}_{\omega^{wh}_5=PE}}{\lambda^{wh}_{\omega^{wh}_6=HD}}+1)}.start = \;\; 14Seconds$$
$$Where \;\; 1 \leq j \leq \lambda^{wh}_{\omega^{wh}_6=HD} \;\; and$$
$$(\frac{\lambda^{wh}_{\omega^{wh}_5=PE}}{\lambda^{wh}_{\omega^{wh}_6=HD}}) * j \leq \lambda^{wh}_{\omega^{wh}_5=PE} \;\; and \;\; \lambda^{wh}_{\omega^{wh}_5=PE} \mod \lambda^{wh}_{\omega^{wh}_6=HD} = 0 \tag{6.46}$$

6.2 Experimental Results

Our aim from this experiment is to validate our theory by identifying different test cases. Run these test cases through the actual system. Then compare these results with the results we get when we run same cases through our model.

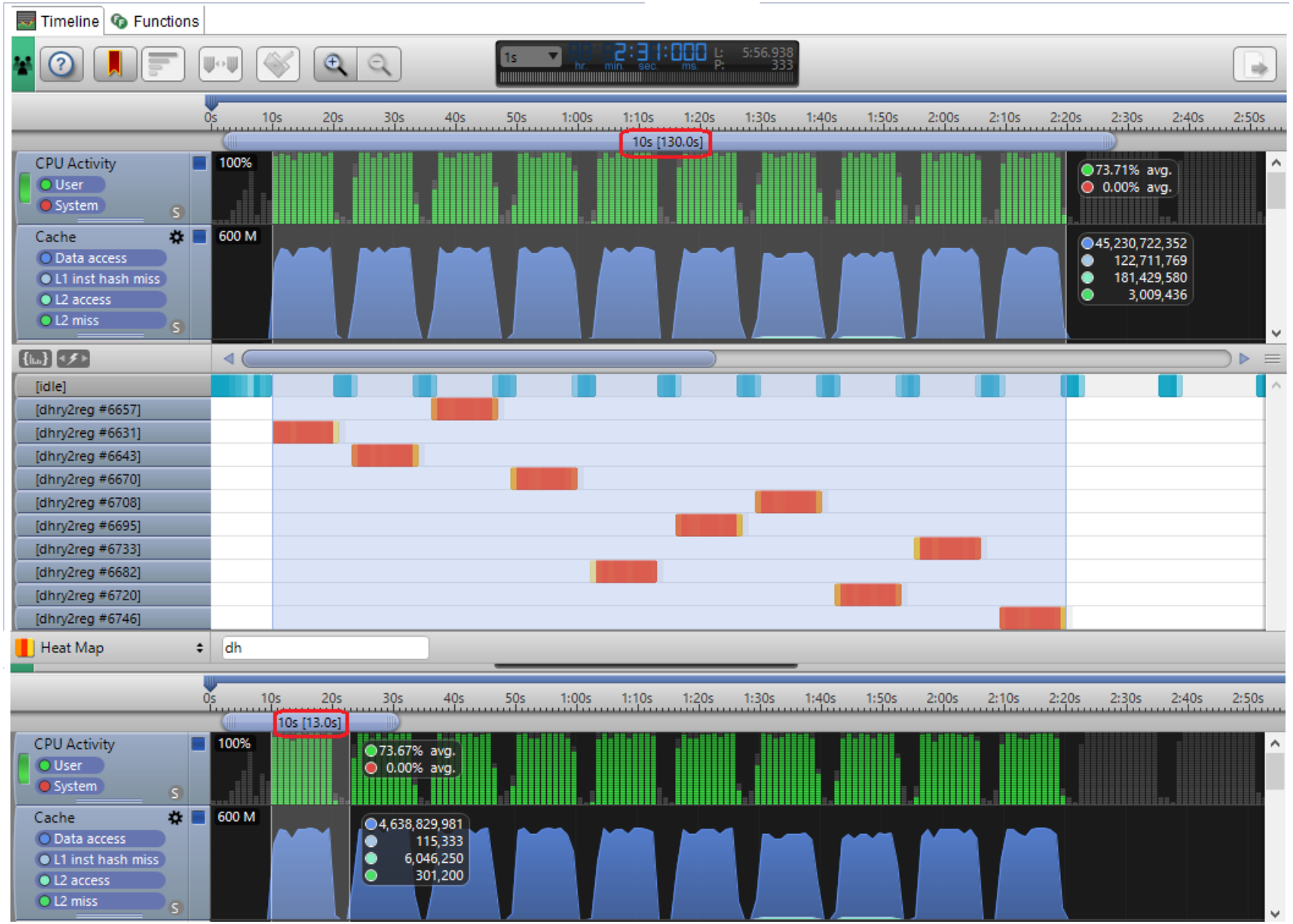

Figure 6.3 Streamline Ds-5 Dhrystone application.

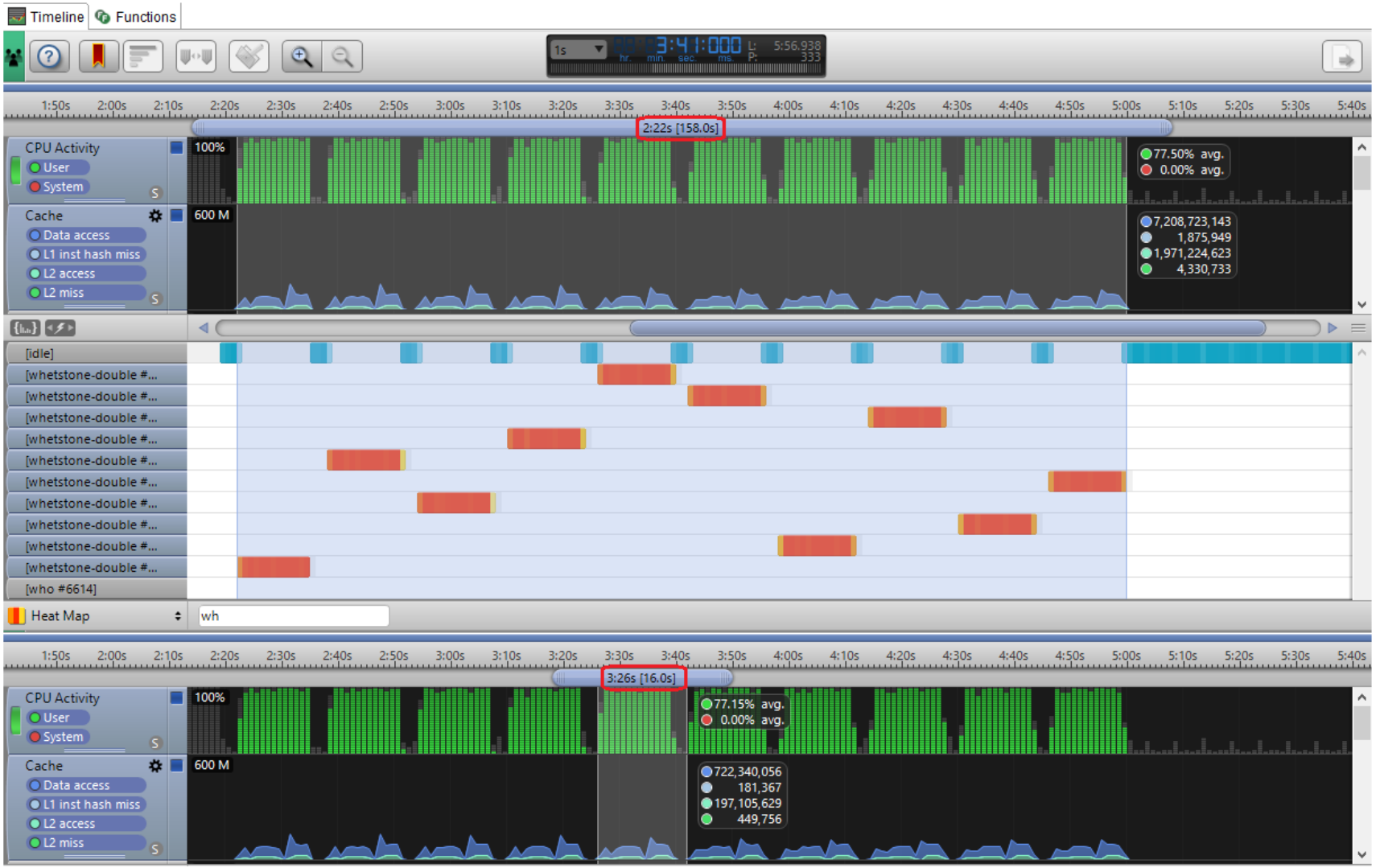

Figure 6.4 Streamline DS-5 WhestStone application.

All experiments were run on an Intel Core i7 computer with 8 GB RAM. All data was captured using ARM DS-5 stream line v5.2. And the model was run on IBM ILOG CPLEX Optimization Studio v12.6.1.0.

We defined in Table 6.5 the different parameters considered with each application. Because BeagleBone black processor core has only four counters to be used and we need five, we had to repeat each of the experiments 20 times while replacing the counters used on the DS-5 stream line. Then we used the average in our mode.

The design space is explored by manually and sequentially applying these parameters. Because in the synchronous dataflow (SDF) MoC, the static data rate allows for the construction of periodic schedules with bounded memory size at compile time [43], the simulation deadline defined was considered sufficient to ensure the validity of an infinite behavior of the system.

We tested our model with different combinations of 16 different application configurations mentioned earlier. The results came to confirm our theory, and the model proposed earlier.

Results are shown in Table 6.3. We used three different terms to represent the results for each case: "success" terms under (Run column) indicates that the application run successfully in the system, "Fail" on the other hand, indicates that the system will not successfully run the application. (Model time) column indicates the time in seconds taken by the CSP model to produce a successful schedule for the tasks. "—" indicates the solver could neither find a feasible schedule nor prove that there is none for this bandwidth within a 10-minute time limit. The (System time) is the time the application actually took to run on the system. Note that we did not consider the time needed by the operating system to start or to run the application which should be between two and 10 minutes. The (Results) column indicates if the CSP model results match the system running results.

Results in Table 6.3 show a comparison between the time taken by the CSP model to identify a decision to the actual running time into the system. The x-axis shows the different cases, and the Y-axis shows the time in seconds. We can see that the CSP model typically

Table 6.3 Single case running test results. * indicates further explanation in the results discussion.

Case	Run	Model time	System time	Results
a (c1)	Success	6.7s	77s	Match
a (c3)	Success	3.92s	77s	Match
b (c2)	Success	3.73s	120s	Match
b (c4)	Fail	1.50s	120s	Match
c	Success	3.47s	77s	Match
d	Success	3.51s	172s	Match
e (c1)	Success	3.65s	77s	Match
e (c3)	Success	3.29s	77s	Match
f (c2)	Success	3.50s	120 s	Match
f (c4)	Fail	1.42s	120 s	Match
g (c1)	Success	4.21s	77 s	Match
g (c3)	Success	4.10s	77 s	Match
h (c2)	Success	3.55s	120 s	Match
h (c4)	Fail	2.55s	120 s	Match
i	Success	3.87s	77 s	Match
j	Success	3.34s	172 s	Match
k (c1)	Success	3.24s	77 s	Match
k (c3)	Success	3.46s	77 s	Match
l (c2)	Success	2.86s	120 s	Match
l (c4)	Fail	2.46s	120 s	Match
m	$Success^*$	1.96s	1s * noF	$Match^*$
n	$Fail^*$	1.96s	5s * noW + 15s	$Not - Match^*$
o	Success	2.32s	130 s	Match
p	Success	3.11s	160 s	Match

takes a few seconds whereas the system takes a few minutes to get the decision. We only consider the application running time in the system.

All the test cases match the actual run except for cases m and n. In case m, where we represent downloading files, the actual system fails to respect the download deadline after the ninth file. On the other hand, the CSP model detects this failure early on the seventh file. Although this is not a match it is still compatible since the CSP model is more restricted. It did not wrongly attribute success.

In the case m the model (Where Success noted with a $*$ in Table 6.3) as explained earlier in the previous paragraph, the system was able to to download the files up to nine files each in one second with separation one second between the start of two sequential downloads. Each is 2MB and the network download speed is 25 Mbit/Second. With this download speed the system should be able to download more files within 9 seconds. But the system failed after the ninth file. It is interesting because the failure occurs with the file download was not expected but it happens. This means it is due to some system limitation not network speed limitation which is the more expected reason. The CSP model was able to detect the same failure but earlier (only at the seventh file).

The false success occurring in case n can be explained. Consider the two figures 6.5 and 6.6 taken from streamline analysis. The first page shows the results of browsing web pages using ice weasel separately on a separate web browseres or pages by closing one browser before opening the next. Following this way the test would work fine and be consistent with the CSP model. But this is not the case since usually we open a new page in a new tab in the same browser and maybe not close the previous one. With ice weasel even if you closed the old tab the system halted after the eleventh tab and the CPU got full contention (Figure 6.6). This irregular behavior could not be represented in the CSP model. This is because even when you close the current tab the application will be still loaded in the memory causing activities not being completely closed.

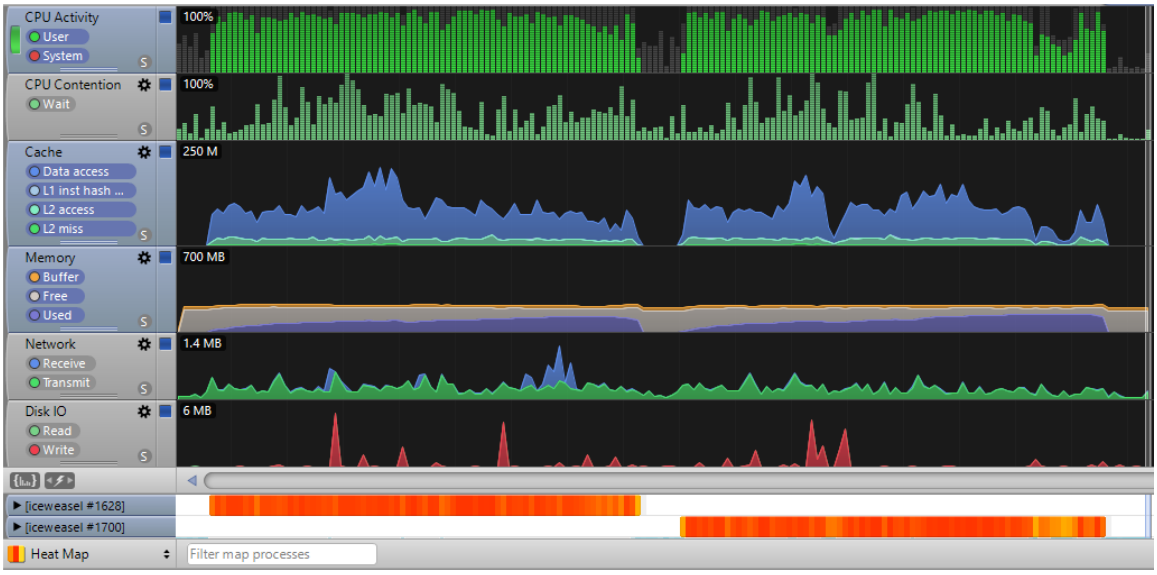

Figure 6.5 Streamline for ice weasel applications browsing multiple web pages in a separate window.

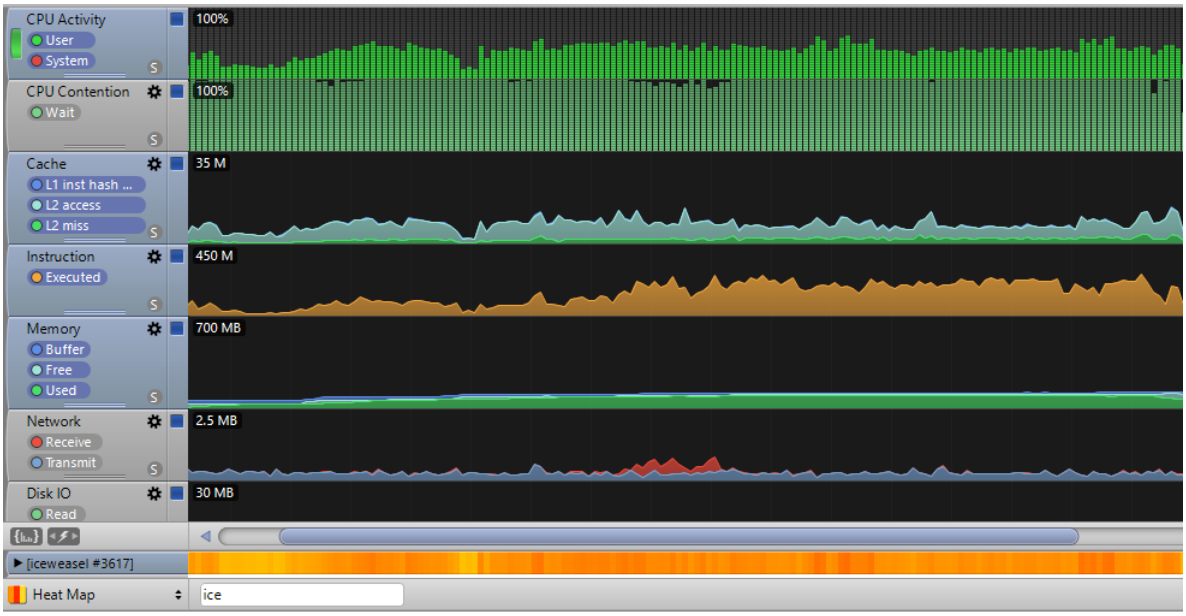

Figure 6.6 Streamline for ice weasel application browsing multiple web pages in a separate tab.

Table 6.4 different case combinations running test results. * indicates further explanation in the results discussion.

Case	Wg				Dh				Wh			
	Run	**Model time**	**System time**	**Results**	**Run**	**Model time**	**System time**	**Results**	**Run**	**Model time**	**System time**	**Results**
a (c1)	S	2.7s	77s	*Match**	S	2.21s	130s	**Match**	S	1.58s	160s	**Match**
a (c3)	S	3.2s⊛	77s	*Match**	S	2.14s	130s	**Match**	S	2.18s	160s	**Match**
b (c2)	S	3.51s	120s	*Match**	S	2.85s	130s	**Match**	S	2.98s	160s	**Match**
c	S	2.71s	77s	*Match**	S	2.21s	130s	**Match**	S	2.32s	160s	**Match**
d	S	3.62s	172s	*Match**	S	2.21s	172s	**Match**	S	2.89s	172s	**Match**
e (c1)	S	1.68s	77s	*Match**	S	2.21s	130s	**Match**	S	1.58s	160s	**Match**
e (c3)	S	2.43s⊛	77s	*Match**	S	3.21s	130s	**Match**	S	4.11s	160s	**Match**
f (c2)	S	2.54s	120s	*Match**	S	2.76s	130s	**Match**	S	3.16s	160s	**Match**
g (c1)	S	4.21s	77s	*Match**	S	2.41s	130s	**Match**	S	3.58s	160s	**Match**
g (c3)	S	2.25s⊛	77s	*Match**	S	3.71s	130s	**Match**	S	3.68s	160s	**Match**
h (c2)	S	3.15s	120s	*Match**	S	5.07s	130s	**Match**	S	2.98s	160s	**Match**
i	S	3.77s	77s	*Match**	S	3.21s	130s	**Match**	S	4.10s	160s	**Match**
j	S	2.57s	172s	*Match**	S	2.26s	130s	**Match**	S	3.10s	160s	**Match**
k (c1)	S	2.47s	77s	*Match**	S	4.12s	130s	**Match**	S	3.87s	160s	**Match**
k (c3)	S	3.15s⊛	77s	*Match**	S	3.21s	130s	**Match**	S	2.66s	160s	**Match**
l (c2)	S	2.86s	120s	*Match**	S	2.71s	130s	**Match**	S	2.92s	160s	**Match**

Case	Application name	Application Parameters
a	$Al + Vl$	Al. Duration = Vl. Duration = 77 s, Al. Tduration = Vl. Tduration = 1 s, Al. startDelay = Vl.startDelay= 5 s, Activity type = entertainment, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample) c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
b	$Al + Vl$	Al. Duration = Vl. Duration = 120 s, Al. Tduration = Vl. Tduration = 1 s, Al. startDelay = Vl.startDelay= 5 s, Activity type = news, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample) c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps,
c	Al	Al. Duration = 77 s, Al. Tduration = 1 s, startDelay = 5 s, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample)
d	Al	Al. Duration = 172 s, Al. Tduration = 1 s, startDelay = 5 s, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample)
e	Vl	Vl. Duration = 77 s, Vl. Tduration = 1 s, Vl.startDelay= 5 s, Activity type = entertainment c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
f	Vl	Vl. Duration = 120 s, Vl. Tduration = 1 s, Vl.startDelay= 5 s, Activity type = news c1: Video resolution = 314 *240 24bpp 30 fps,

Continued on next page

Table 6.5 – continued from previous page

Case	Application name	Application Parameters
		c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
g	$An + Vn$	An.Duration = Vn. Duration = 77 s, An.Tduration = Vn. Tduration = 1 s, An. startDelay = Vn.startDelay= 10 s, Activity type = entertainment, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample), Download speed = 25 Mbps, c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
h	$An + Vn$	An.Duration = Vn. Duration = 120 s, An.Tduration = Vn. Tduration = 1 s, An. startDelay = Vn.startDelay= 10 s, Activity type = news, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample), Download speed = 25 Mbps, c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
i	An	An. Duration = 77 s, An.Tduration = 1 s, An.startDelay = 5 s, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample), Download speed = 25 Mbps
j	An	Al. Duration = 172 s, Al. Tduration = 1 s, startDelay = 5 s, Audio output = 44100 Hz 2 ch floatle (4 bytes per sample), Download speed = 25 Mbps

Continued on next page

Table 6.5 – continued from previous page

Case	Application name	Application Parameters
k	Vn	Vn. Duration = 77 s, Vn. Tduration = 1 s, Vn.startDelay= 10 s, Activity type = entertainment, Download speed = 25 Mbps, c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
l	Vn	Vn. Duration = 120 s, Vn. Tduration = 1 s,Vn.startDelay= 10 s, Activity type = news, Download speed = 25 Mbps, c1: Video resolution = 314 *240 24bpp 30 fps, c2: Video resolution = 426 *240 24bpp 30 fps, c3: Video resolution = 470 *360 24bpp 30 fps, c4: Video resolution = 1280 *720 24bpp 30 fps
m	Wg	Wg. Duration = 1 s, Wg. Tduration = 1 s, Wg.startDelay= 1 s, Gap between files = 1, Download speed = 25 Mbps, $1 \leq noF \leq 20$
n	Iw	Iw. Duration = 40 s, Iw. Tduration = 1 s, Iw.startDelay= 5 s, Gap between webs = 5, Download speed = 25 Mbps, $1 \leq noW \leq 20$
o	Dh	Dh. Duration = 130 s, Dh. Tduration = 13 s, Dh. start-Delay = 1 s
p	Wh	Wh. Duration = 160 s, Wh. Tduration = 16 s, Wg. startDelay = 1 s

Table 6.5: Summary of the Different Parameters Considered for Each Application

The next step in our experiment is to test different application combinations. As the most load on the system comes from stream applications so we tested running these applications when combined with Wg, Iw, Dh, and Wh. Table 6.4 shows the results of different combinations. Note that although we used the same notations from Table 6.3, we replaced

"Success" with a "S" symbol to save space.

We did not go through with combining with the Iw application because of its original results when we tested it running stand-alone. We did not expect accurate results for such a combination either. Furthermore we excluded cases $c(c4)$, $f(c4)$, $h(c4)$, and $l(c4)$. Theses four cases were failed running as stand-alone applications. So there is no need to test their combinations as it is expected to fail.

A simple direct result came from combining test cases with Dh and, Wh applications. As we can see, both applications had no effect on the system and considered low priority on the processor. It created a Match/Success in all cases. At the same time, we can see the big gap in time detection between CSP model and actual run.

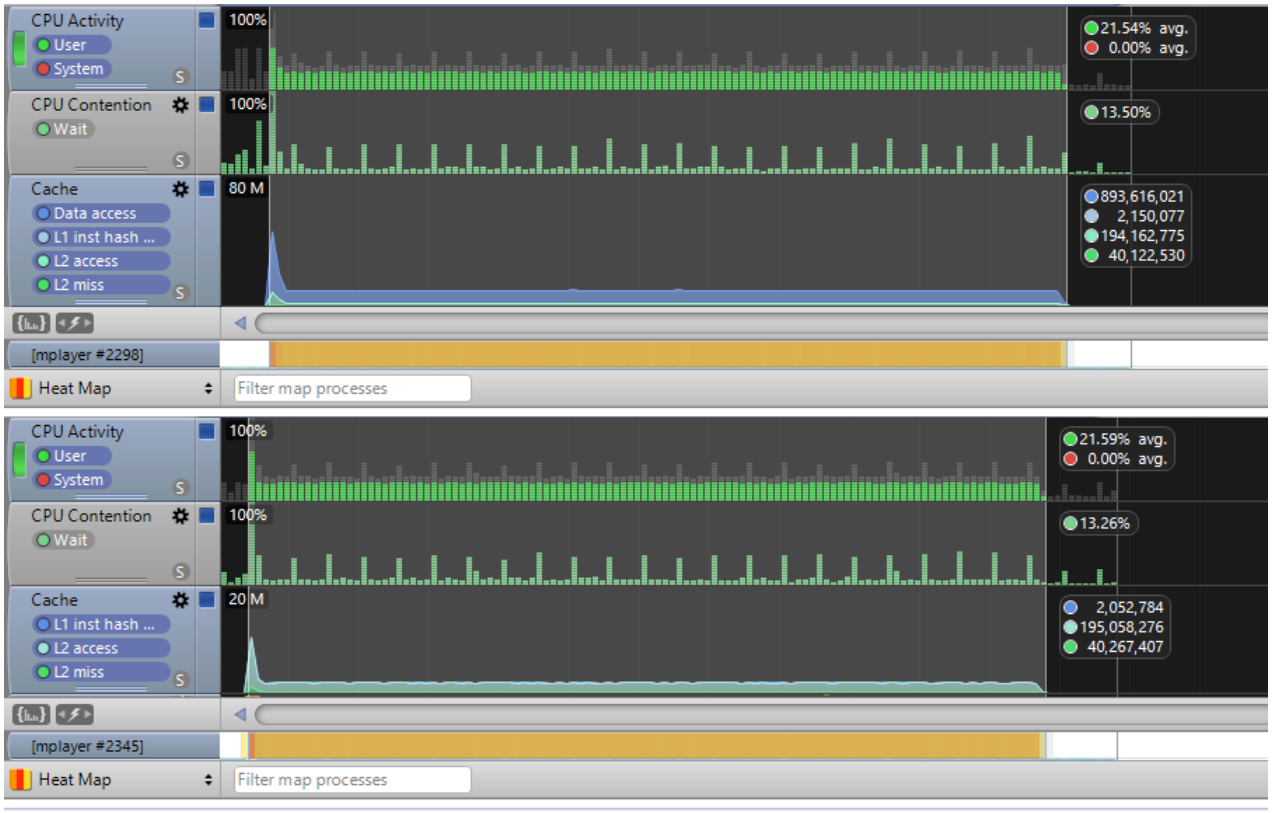

Figure 6.7 Two different streamline analysis for MPlayer applications running local video with low decompression rate for newsletter program (Case $b(c2)$).

However the Wg application has a different result. We can combine the explanation of

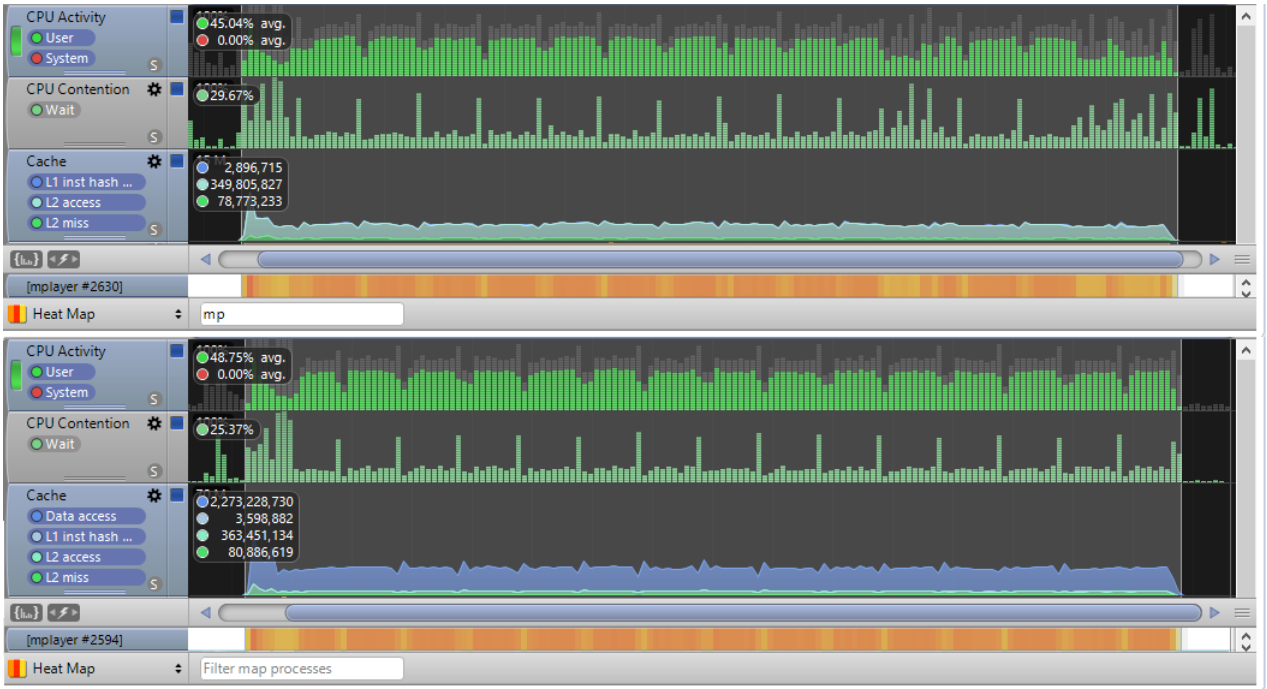

Figure 6.8 Two different streamline analysis for MPlayer application running local video with high decompression rate for football game (Case $a(c1)$).

some of the results. First we would like to comment on the * beside the Match in Table 6.4. It means that we got a match in the results between the CSP model and the actual system running but with a different condition associated with the application:

- Cases $a(c3), e(c3), g(c3), andk(c3)$: system failure was detected in the CSP model when noF for the Wg application reached six files. However the actual failure occurs at the eighth file.
- All other cases: system failure detected in the CSP model when noF for the Wg application reached seven files. However the actual failure occurs at the eighth file.

In both cases the CSP model was more restricted than the actual run (it detected failure with Wg using a less noF number then the actual run. This means less restrictions and resources used in the system) which means that the CSP model did not detect a wrong success but instead a wrong failure. This is less critical.

Secondly, we will comment on $\circledast$ beside the Model time column in Table 6.4. Although having a time in this field means the solver successfully found a solution, this was not the

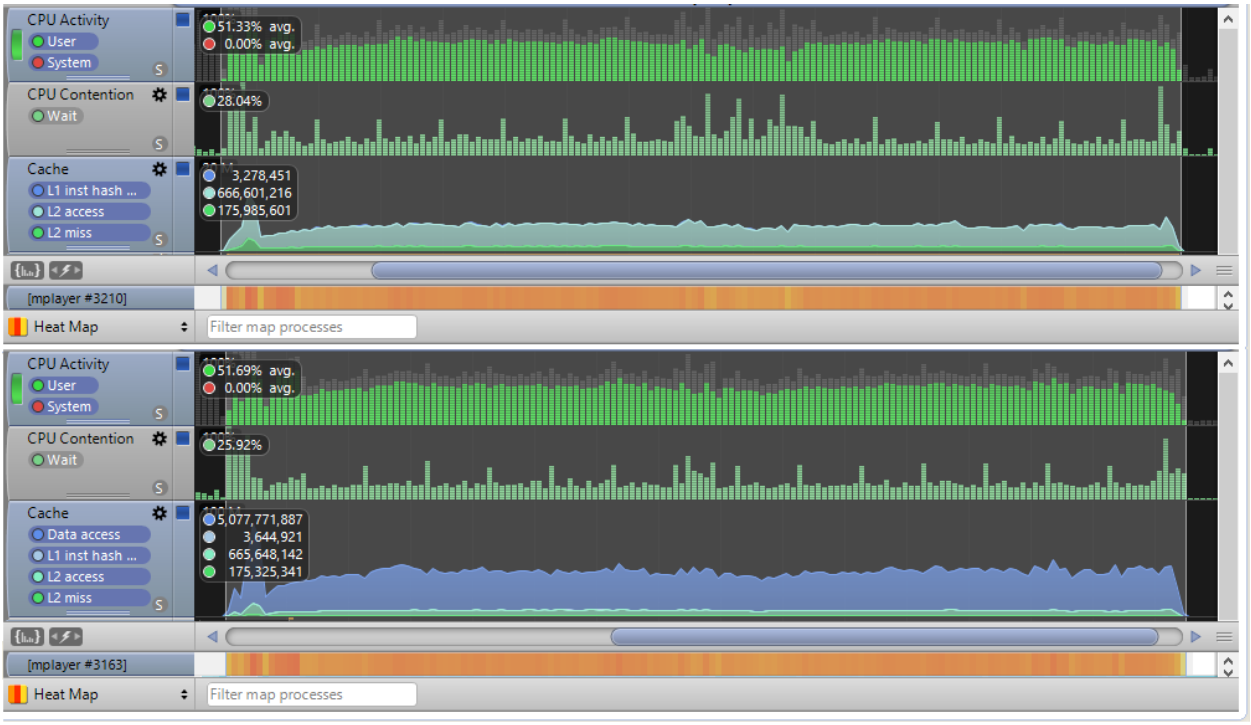

Figure 6.9 Two different streamline analysis for MPlayer application running live stream video with high decompression rate for football game (Case $g(c1)$).

case all the time. For the four cases $a(c3), e(c3), g(c3), andk(c3)$ as previously mentioned we test the combination with Wg with different numbers of files noF. Although we detected failure on the sixth file and success on one, two, and three files, still the fourth and fifth file cases were infeasible. The CSP model still could not find a solution for these two cases.

Looking at the bigger picture for the complete results combining with Wg applications, the result shows much better timing with not much difference in the results.

By analyzing our results, some cases give more interesting information about the system running:

1. Cases $a(c3), e(c3), g(c3), andk(c3)$, local and live stream video running with high decompression rate and big frame size, show the system would be affected by other applications running Wg and reduce the efficiency of this application to keep its initial performance. We can note that there was not much different in performance between live stream and local video. This is supported by the analysis we get from DS-5 streamline. Comparing performance is shown in Figures 6.8 and 6.9. The two figures show the real-time analysis of local and live stream videos running in the system. The CPU activities for both applications were almost the same (Local 45%:48%, Live 51%:52%).
2. Cases $a(c1)$, $e(c1)$, $g(c1)$, and $k(c1)$, local and live stream video running with high decompression rate and smaller frame size, shows no effect on the system even when combined with Wg applications. Combining this with the first note, we can confirm that a not so much change in the frame size could affect the system performance. In this case the frame size reduced from (470 * 360) to (314 * 240).
3. Cases c, d, i, and j, local and live stream audio, shows no effect on the system even when combined with Wg applications. Combining this with the first notes, we can confirm that the video decompression rate has a greater effect on the system performance than audio.
4. Cases $b(c2)$, $f(c2)$, $h(c2)$, and $l(c2)$, local and live stream video running with low decompression rate and bigger frame size, shows no effect on the system even when combined with Wg applications. Although there is a big change in the frame size between first case and this one, still the system did not fail. This is due to the fact that this video has a lower decompression rate. This is supported by the analysis we get from DS-5 streamline. Comparing performance is shown in Figures 6.8 and 6.7. The two figures show the real-time analysis of local stream videos running in the system. The first for a video with high decompression rate and the other for a low decompression rate.

Although the second case shows a bigger frame size display rate ([314 * 240] to [426 * 240]), still it shows much less CPU activity (High 45%-48%, low 21%-22%).

According to the previous results, the CSP model results were consistent with the real system results. It generated decisions in much less time and respected both system and application rules.

CHAPTER 7 CONCLUSION

7.1 Work Summary

This thesis has been concerned with system-level verification using constraint programming. Our focus was to identify the critical system parameters (e.g. buffer size) that can lead to unsatisfied application constraints. We also propose design optimization (e.g. buffer minimization) to increase the system efficiency and reduce its cost. The research emphasizes test-case generation with a particular focus on the interaction between system components, concurrency and resource competition, and the role of running applications in the verification process.

We studied the possibility of creating a system-level scenario that takes into account the system level work-flow with respect to system resources and performance requirements, namely task deadlines, response time, CPU and memory usage, and buffer size. Specifically, we investigated whether the behaviour of different interactions among system components executing different tasks can be effectively re-expressed as a constraint-based scheduling problem over the space of possible inputs to the system, finding out whether we can address similar cases of failure using this model. Solving this problem means finding a better way to investigate early on the system under verification and potentially address a certain case of failure in a very early design stage in a much more reasonable time.

The approach was tested with various applications, different input streams and different architectures. Results show that the methodology is able to identify system failure conditions in a fraction of the time needed by simulation-based verification. It gives the Test Engineer the ability to explore the design space and deduce the best policy, and also helps choose a proper a recommended architecture for the applications running. Our work culminated in modelling an existing architecture on the market running selected applications and comparing our model results with the results coming from running the actual applications on the system.

We do not claim that our work can handle any situation but we show a valid approach with very acceptable results. The model we proposed can be easily changed to test different system architectures running a various applications.

7.2 Future Work

This thesis has a number of issues that should be addressed in our future work:

- **Introduce a work specific search strategy and propagation algorithms that improve results.** The time needed by CP to find a solution depends on three parameters: the size of the search space, the capability to trim down that search space, and the way it is searched. In our model we have two types of DUT. The first, where it is an optimization problem when we try to defined the minimum requirements to run a certain set of applications. This applies to the first two motivated examples mentioned in Chapters 4 and 5. The second type is when we have a specific architecture that cannot be modified and needs to be tested against a number of applications. This applies to the third motivated example mentioned in Chapter 6. To solve our scheduling problem, different basic search strategies can be used:

  - **first-fail strategy:** This strategy can be used with the first type of DUT. We start by assigning the minimum possible values for the used resources. It is based on the principle of beginning where a failure is more likely to be detected. More parts of the search space are removed earlier, and important decisions are taken as soon as possible, rather than waiting for an extensive search to be completed.

  - The second search strategy is similar to the previous one. But it can be used with the two types of DUT. It is based on application needs constraints. We start by choosing the task with the biggest resource need or hardest deadline.

- **Create a variety of test cases:** although the third motivated example had different types of applications, these applications were not fully investigated. As the testbench provides a powerful observation tool for both the DUT and the application, we need to give more space to the testbench used, run it for longer periods of time, and give it a higher priority on the system. We would study how this should affect the results. For example, in the third motivated example we used only two tests (Dhrystone, Whetstone) out of the ten available with unixBench testbench (execl Throughput, File Copy, Pipe Throughput, Pipe-based Context Switching, Process Creation, Shell Scripts, System Call Overhead, Graphical Tests). Each of the two tests ran only once. This was not enough to generate enough load to the system, to cover more cases. Considering more tests in different timing and order will increase

the varity and quality of the generated test cases.

- **Insure Portability among different DUT specifications with minimum modification to the model used:** In our third motivated example we used DS-5 to measure the performance of each of the applications used to create scenarios running through the CSP model. Although the model was able to predict the same results in a shorter period of time still, the model needed the information collected from the actual application running in the system. This makes the CSP model need more time than the actual test time. To argue with that we need to test these scenarios with other platforms without the need to first recover information about the application traffic and performance on the new platform using DS-5. We still have to prove the model will provide acceptable results with the same application properties and information collected from the original platform.

- **Provide further analysis of applications' irregular behaviour:** As we show in our third motivated example we received irregular behaviour of false success with the Ice Weasel web browser. This behaviour is very interesting and needs further analysis after adding more constraints to model this behaviour.

REFERENCES

[1] Adir, A. and Almog, E. and Fournier, L. and Marcus, E. and Rimon, M. and Vinov, M. and Ziv, A. (2004). Genesys-pro: Innovations in test program generation for functional processor verification. *Design & Test of Computers, IEEE*, *21*(2), 84–93.

[2] Adir, Alion and Bin, Eyal and Peled, Ofer and Ziv, Avi (2003). Piparazzi: a test program generator for micro-architecture flow verification. *Proceeding of the Eighth IEEE International Workshop on High-Level Design Validation and Test (HLDVT 03)*. IEEE Press, 23–28.

[3] Adir, Allon and Emek, Roy and Katz, Yoav and Koyfman, Anatoly (2003). Deeptrans-a model-based approach to functional verification of address translation mechanisms. *Proceeding of the fourth IEEE International Workshop on Microprocessor Test and Verification: Common Challenges and Solutions*. IEEE Press, 3–6.

[4] Aharoni, M. and Asaf, S. and Fournier, L. and Koifman, A. and Nagel, R. (2003). Fpgen-a test generation framework for datapath floating-point verification. *Proceeding of the Eighth IEEE International Workshop on High-Level Design Validation and Test (HLDVT03)*. IEEE, 17–22.

[5] Barbieri, Roberto and Bruschi, Danilo and Rosti, Emilia (2002). Voice over ipsec: Analysis and solutions. *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC'02)*. IEEE Press, 261–270.

[6] Beltrame, G. and Fossati, L. and Sciuto, D. (2009). Resp: a nonintrusive transaction-level reflective mpsoc simulation platform for design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *28*(12), 1857–1869.

[7] Benini, L. and Lombardi, M. and Milano, M. and Ruggiero, M. (2008). A constraint programming approach for allocation and scheduling on the cell broadband engine. *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*. Springer LNCS 5202, 21–35.

[8] Berry, G. and Blanc, L. and Bouali, A. and Dormoy, J. (2002). Top-level validation of system-on-chip in esterel studio. *Proceeding of the Seventh IEEE International Workshop on High-Level Design Validation and Test Workshop (HLDVT02)*. IEEE, 36–41.

[9] Bohizic, Theodore J and Duale, Ali Y and Wittig, Dennis W (2011). System for estimating and improving test case generation. US Patent 7,904,270.

[10] Bonfietti, A. and Lombardi, M. and Benini, L. and Milano, M. (2012). Global cyclic cumulative constraint. *Proceedings of Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (CPAIOR12)*, 81–96.

[11] Chuah, Chen-Nee and Katz, Randy H (1999). *Network provisioning and resource management for IP telephony*. University of California, Berkeley, Computer Science Division, Report No. UCB/CSD-99-1061.

[12] Coley, Gerald (2013). Beaglebone black system reference manual.

[13] Cplex, IBM Ilog (2010). 12.1 reference manual. *URL {http://www. ilog. com}*.

[14] Demiriz, Ayhan and Bagherzadeh, Nader and Alhussein, Abdulaziz (2015). Using constraint programming for the design of network-on-chip architectures. *Computing*, *97*(6), 579–592.

[15] Ekelin, Cecilia and Jonsson, Jan (2000). Solving embedded system scheduling problems using constraint programming. *Technical report, Dept. of Computer Engineering, Chalmers University of Technology*.

[16] El-Mahi, O. and Nicolescu, G. and Pesant, G. and Beltrame, G. (2012). Embedded system verification through constraint-based scheduling. *The 17th IEEE International High Level Design Validation and Test Workshop (HLDVT12)*.

[17] El-Mahi, Olfat and Pesant, Gilles and Nicolescu, Gabriela and Beltrame, Giovanni (2013). Embedded system verification through constraint-based scheduling. *Proceeding of International Symposium on the Rapid System Prototyping (RSP13)*. IEEE Press, 73–79.

[18] Eles, P. and Doboli, A. and Pop, P. and Peng, Z. (2000). Scheduling with bus access optimization for distributed embedded systems. *Proceeding of the IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *8*(5), 472–491.

[19] Emek, R. and Jaeger, I. and Naveh, Y. and Bergman, G. and Aloni, G. and Katz, Y. and Farkash, M. and Dozoretz, I. and Goldin, A. (2002). X-gen: A random test-case generator for systems and socs. *Proceedings of the Seventh IEEE International Workshop on High-Level Design Validation and Test (HLDVT02)*. IEEE, 145–150.

[20] Fournier, Laurent and Arbetman, Yaron and Levinger, L (1999). Functional verification methodology for microprocessors using the genesys test-program generator. application to the x86 microprocessors family. *Proceedings of the Design Automation and Test in Europe Conference and Exhibition (DATE99)*. IEEE, 434–441.

[21] Govindarajan, R. and Gao, G.R. and Desai, P. (2002). Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks. *The Journal of VLSI Signal Processing*, *31*(3), 207–229.

[22] Gratch, Jonathan and Chien, Steve (1996). Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research*, 365–396.

[23] Haskell, BG and Puri, A. (2012). Mpeg video compression basics. *The MPEG Representation of Digital Media*, 7–38.

[24] Hladik, P.E. and Cambazard, H. and Déplanche, A.M. and Jussien, N. (2008). Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, *81*(1), 132–149.

[25] Hunt, Warren A (2002). Introduction: Special issue on microprocessor verifications. *Formal Methods in System Design*, *20*(2), 135–137.

[26] IBM (2012). *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*. IBM, http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud, 12th édition.

[27] Kumar, Vipin (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, *13*(1), 32.

[28] Laborie, Philippe and Godard, Daniel (2007). Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, 276–284.

[29] Lee, E.A. and Messerschmitt, D.G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, *100*(1), 24–35.

[30] Lombardi, M. and Milano, M. (2012). Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *In Constraints*, *17*(1), 1–35.

[31] Nahir, A. and Ziv, A. and Emek, R. and Keidar, T. and Ronen, N. (2006). Scheduling-based test-case generation for verification of multimedia socs. *In Proceedings of the 43rd Design Automation Conference. New York: Association for Computing Machinery.* ACM Press, 348–351.

[32] Naveh, Yehuda and Richter, Yossi and Altshuler, Yaniv and Gresh, Donna L and Connors, Daniel P (2007). Workforce optimization: Identification and assignment of professional workers using constraint programming. *IBM Journal of Research and Development*, *51*(3.4), 263–279.

[33] Palnitkar, Samir (2004). *Design Verification with e*. Prentice Hall Professional.

[34] Rashinkar, P. and Paterson, P. and Singh, L. (2001). *System-on-a-chip verification: methodology and techniques*. Springer.

[35] Rashinkar, Prakash and Paterson, Peter and Singh, Leena (2007). *System-on-a-chip verification: methodology and techniques*. Springer Science & Business Media.

[36] River, Wind (2006). Simics full system simulator.

[37] Smith, Taber H and White, David (2011). Electronic design for integrated circuits based on process related variations. US Patent 7,962,867.

[38] Stuijk, S. and Basten, T. and Geilen, MCW and Corporaal, H. (2007). Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. *Proceedings of the 43rd annual conference on Design automation (DAC'07)*. IEEE Press, 777–782.

[39] Van Hentenryck, P. and Perron, L. and Puget, J.F. (2000). Search and strategies in opl. *ACM Transactions on Computational Logic (TOCL)*, *1*(2), 285–320.

[40] Wang, Rui and Zhan, Wenfa and Jiang, Guisheng and Gao, Minglun and Zhang, Su (2004). Reuse issues in soc verification platform. *Proceedings of The 8th International Conference on Computer Supported Cooperative Work in Design*. IEEE, vol. 2, 685–688.

[41] Wolf, W. (2004). The future of multiprocessor systems-on-chips. *Proceedings of the 41st annual Design Automation Conference*. ACM, 681–685.

[42] XU, Xiao Xi (2009). *Software-Centric and Interaction-Oriented System-on-Chip Verification*. Thèse de doctorat, Shanghai Jiao Tong University, China, 1996.

[43] Zhu, Jun and Sander, Ingo and Jantsch, Axel (2008). Energy efficient streaming applications with guaranteed throughput on mpsocs. *Proceedings of the 8th ACM international conference on Embedded software*. ACM, 119–128.

[44] Zhu, J. and Sander, I. and Jantsch, A. (2009). Buffer minimization of real-time streaming applications scheduling on hybrid cpu/fpga architectures. *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 1506–1511.

[45] Zhu, J. and Sander, I. and Jantsch, A. (2010). Constrained global scheduling of streaming applications on mpsocs. *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. IEEE Press, 223–228.